

HELP DOCUMENTATION

# USER MANAGEMENT RESOURCE ADMINISTRATOR 10.8

tools4ever



## Contents

<b>1.</b>	<b>Welcome to UMRA</b>	<b>1</b>
<b>2.</b>	<b>Release notes</b>	<b>2</b>
2.1.	User Management Resource Administrator release notes .....	2
2.2.	Upgrading MASS projects from older versions .....	50
<b>3.</b>	<b>UMRA User Guide</b>	<b>1</b>
3.1.	UMRA Basics .....	3
3.1.1.	Introduction .....	3
3.1.2.	UMRA scripting .....	5
3.1.3.	UMRA Projects .....	9
3.1.4.	UMRA components .....	20
3.1.5.	UMRA project management .....	1
3.2.	Getting Started .....	3
3.2.1.	Introduction .....	4
3.2.2.	Mass updating network resources - Mass module .....	1
3.2.3.	Delegating user account management tasks - Forms module .....	4
3.2.4.	.....	4
3.2.5.	User account provisioning - Automation module .....	7
3.2.6.	Installing UMRA .....	9
3.2.7.	Creating a MASS example project - Mass create users .....	12
3.2.8.	Creating a Forms example project - Reset password .....	32
3.2.9.	Appendix A - Script actions .....	64
3.2.10.	Appendix B - Installing the UMRA Service .....	69

---

3.3.	Integrate UMRA with other applications using COM .....	75
3.3.1.	UMRA COM.....	1
3.3.2.	UMRA COM objects .....	3
3.3.3.	UMRA COM in VB scripts .....	33
3.3.4.	UMRA COM in IIS .....	47
3.3.5.	References .....	74
3.4.	Managing printer queues.....	77
3.4.1.	Introduction .....	1
3.4.2.	UMRA projects for managing printer queues.....	1
3.5.	Managing Windows computer services.....	21
3.5.1.	Project definition .....	1
3.5.2.	Project structure .....	1
3.5.3.	Step 1: Environment setup .....	2
3.5.4.	Step 2: Form project - Collect Services .....	3
3.5.5.	Step 3: Form project - Manage Services.....	6
3.5.6.	Project execution.....	21
3.5.7.	Contacts .....	23
3.6.	Managing LDAP directory services using UMRA.....	25
3.6.1.	Introduction .....	1
3.6.2.	Concept.....	2
3.6.3.	UMRA LDAP script actions .....	5
3.6.4.	Directory Service tasks.....	15
3.6.5.	Novell eDirectory .....	19
3.6.6.	Linux OpenLDAP .....	57
3.6.7.	Microsoft Active Directory.....	73
3.6.8.	References .....	120

---

3.7.	Name generation .....	121
3.7.1.	Generating user names.....	1
3.8.	UMRA tables .....	9
3.8.1.	Introduction .....	1
3.8.2.	The concept of tables in UMRA .....	3
3.8.3.	Special table type - Generic table Variable.....	16
3.8.4.	Processing user input.....	20
3.8.5.	Formatting tables.....	23
3.8.6.	Using tables in UMRA - Forms & Delegation - Hands-on .....	24
3.8.7.	Contacts .....	36
3.9.	Lotus Notes user guide .....	36
3.9.1.	Configuring the UMRA console for use with Lotus Notes .....	37
3.9.2.	Configuring the UMRA service for use with Lotus Notes .....	42
3.9.3.	Administration Requests database.....	47
3.9.4.	Lotus Notes example projects .....	54
3.10.	Exchange 2007 .....	58
3.10.1.	Introduction Exchange 2007 .....	1
3.10.2.	Requirement UMRA Exchange 2007 support .....	1
3.10.3.	Manage Active Directory with the UMRA Powershell Agent service .....	2
3.10.4.	Managing Exchange 2003 with the UMRA Powershell Agent service .....	2
3.10.5.	Setting up the Exchange 2007 Management Tools on a 32-bit platform.....	3
3.10.6.	Using the Exchange Web Services with UMRA.....	7
3.11.	Exchange 2010 .....	25
3.11.1.	Introduction Exchange 2010 .....	1
3.11.2.	Accessing Exchange 2010 functionality from an UMRA project.....	2



---

3.12.	Office 365.....	4
3.12.1.	Introduction Office 365.....	1
3.12.2.	Office 365 Users.....	1
3.13.	Powershell Agent service.....	2
3.13.1.	Powershell Agent service.....	2
3.13.2.	UMRA - Powershell Agent service .....	3
3.13.3.	UMRA action - Powershell script conversion.....	3
3.13.4.	UMRA dynamic actions.....	3
3.13.5.	Installation .....	3
3.13.6.	Configuration and settings.....	18
3.13.7.	UMRA dynamic actions.....	22
3.13.8.	Manage Active Directory with the UMRA Powershell Agent service .....	72
3.13.9.	Managing Exchange 2003 with the UMRA Powershell Agent service .....	73
3.13.10.	Setting up the Exchange 2007 Management Tools on a 32-bit platform .....	74
3.13.11.	Powershell Agent service session .....	78
3.13.12.	UMRA Sessions.....	81
3.13.13.	Configuring a secure web-site with IIS.....	83
3.14.	UMRA Google Module .....	100
3.14.1.	Google - Requirements .....	100
3.14.2.	Google - Action: Google Setup Connection .....	101
3.14.3.	Google - Connections.....	103
3.14.4.	Google - Registry settings .....	103
3.15.	UMRA SAP module .....	107
3.15.1.	SAP - Requirements .....	107
3.15.2.	SAP - Action: SAP Setup connection .....	108
3.15.3.	SAP - Connections .....	108

3.15.4.	SAP - UMRA SAP child process.....	109
3.15.5.	SAP - SAP Generic function module.....	109
3.15.6.	SAP - Example projects .....	111
3.15.7.	SAP - Registry settings.....	111
3.16.	AFAS Online .....	115
3.17.	Setup connection.....	115
3.18.	AFAS get employees .....	115
3.19.	AFAS get employees contract.....	116
3.20.	AFAS Get organigram.....	116
3.21.	AFAS Export Date.....	116
3.22.	AFAS Update employee .....	117
3.23.	Password Synchronization Manager.....	118
3.23.1.	Goal.....	119
3.23.2.	Installing UMRA PSM for the first time.....	119
3.23.3.	Miscellaneous UMRA PSM topics .....	124
3.23.4.	Manage Active Directory with the UMRA Powershell Agent service .....	132
3.24.	Education .....	133
3.24.1.	Aura connector installation .....	133
3.25.	SOAP Synchronization template project .....	148
<b>4.</b>	<b>UMRA Reference Guide</b>	<b>1</b>
4.1.	Script action overview.....	3
4.1.1.	User.....	3
4.1.2.	.....	91
4.1.3.	Active Directory .....	117
4.1.4.	Exchange.....	156

4.2.....	162
4.3.....	168
4.3.2. File System.....	341
4.3.3. Other actions .....	369
4.3.4. Windows computer services .....	373
4.3.5. Managing printers and printer queues.....	380
4.3.6. LDAP directory services .....	383
4.3.7. Lotus Notes.....	392
4.3.8. SAP actions .....	488
4.3.9. TOPdesk .....	488
4.3.10. Education .....	490
4.3.11. Variable actions.....	527
4.3.12. Programming .....	567
4.3.13. Mail .....	575
4.3.14. Powershell.....	578

<b>5.</b>	<b>Context sensitive Help</b>	<b>607</b>
5.1.	UMRA PSM Domain Controllers Overview .....	607
5.2.	Installation and upgrade wizard- Installation and upgrade options.....	608
5.3.	Installation and upgrade wizard - Specify the target domain.....	609
5.4.	Installation and upgrade wizard - Specify the target domain controller .....	609
5.5.	Install/upgrade software.....	609
5.6.	Delete Options .....	610
5.7.	Domain Controller Options.....	610
5.8.	Reboot options.....	611
5.9.	Refresh options .....	612
5.10.	Advanced Settings - general settings.....	612
5.11.	Advanced Settings - domains .....	612
5.12.	Select domain controller wizard - Specify the target domain .....	612
5.13.	Select domain controller wizard - welcome .....	613
5.14.	Specify the name of the domain controller.....	613
5.15.	Password Synchronisation Manager service settings.....	613
5.16.	IDD_TAB_ACTIONITEM_LN_QUERY_ITEMS- forwarded .....	613
5.17.	IDD_TAB_ACTIONITEM_LN_ACL -forwarded .....	613
5.18.	IDD_TAB_ACTIONITEM_CYCOS_GET_ATTACHMENT .....	614
5.19.	IDD_TAB_ACTIONITEM_CYCOS_GET_CUSTOM .....	614
5.20.	IDD_TAB_ACTIONITEM_CYCOS_SET_CUSTOM .....	614
5.21.	IDD_DIALOG_CYCOS_CUSTOMFIELD_OUTPUT .....	614
5.22.	Value of text item. ....	614
5.23.	Value of text list item.....	614
5.24.	Value of date-time item.....	615
5.25.	Value of numeric item .....	617

---

5.26.	Built-in variables .....	618
5.27.	Condition criteria - Setup.....	622
5.28.	Condition criteria - Setup criterion.....	623
5.29.	Configure predefined variables .....	623
5.30.	Control running UMRA service projects .....	624
5.31.	Data specification - Text list.....	624
5.32.	Database query - Database specification .....	626
5.33.	Database query - Query.....	626
5.34.	Database setup - MS-Access (Jet).....	627
5.35.	Database setup - Other databases .....	628
5.36.	Expiration date.....	631
5.37.	For each - Input variables .....	632
5.38.	Form action - Check form input.....	634
5.39.	Form action - Execute script of form .....	635
5.40.	Form action - General .....	635
5.41.	Form action - Iteratively execute project script .....	636
5.42.	Form action - Return current form .....	637
5.43.	Form action - Return other form .....	638
5.44.	Form action - Set variable value .....	638
5.45.	Form action - Execute command line at client workstation.....	638
5.46.	Form fields - Button .....	639
5.47.	Form fields - Checkbox .....	640
5.48.	Form fields - Display .....	642
5.49.	Form fields - Input text .....	644
5.50.	Form fields - Name .....	646
5.51.	Form fields - Picture.....	646

---

5.52.	Form fields - Radio button .....	647
5.53.	Form fields - Static text .....	648
5.54.	Form fields - Table - Columns .....	649
5.55.	Form fields - Vertical space .....	651
5.56.	Form fields - Table - Data refresh .....	651
5.57.	Form fields - Table - Exclusions .....	653
5.58.	Form fields - Table - Fixed data .....	654
5.59.	Form fields - Table - Generic table .....	655
5.60.	Form fields - Table - Network call parameters .....	655
5.61.	Form fields - Table - Network table .....	657
5.62.	Form fields - Table - Options .....	659
5.63.	Form fields - Table - Row icon image .....	660
5.64.	Form fields - Table - Type .....	660
5.65.	Form project - Form fields .....	661
5.66.	Function modules .....	663
5.67.	Generic table - Introduction .....	670
5.68.	Generic table - Run test .....	671
5.69.	Generic table - Table type .....	672
5.70.	Generic table - Column names .....	674
5.71.	Generic table - Variable .....	674
5.72.	Interface modules .....	675
5.73.	LDAP attributes - Attribute specification .....	675
5.74.	LDAP attributes - Data conversion .....	679
5.75.	.....	679
5.76.	LDAP attributes - Data conversion routine .....	680
5.77.	LDAP Directory Service - Encrypt input .....	684

---

5.78.	LDAP Directory Service - LDAP Search .....	686
5.79.	LDAP Directory Service - LDAP Search Attributes.....	687
5.80.	LDAP Directory Service - Setup LDAP modification data .....	688
5.81.	LDAP search - Attributes.....	689
5.82.	LDAP search - LDAP binding.....	694
5.83.	LDAP search - LDAP Filter .....	697
5.84.	LDAP search - Options .....	702
5.85.	License code.....	705
5.86.	License model .....	707
5.87.	License matrix.....	709
5.88.	Log information .....	711
5.89.	Manage script actions.....	712
5.90.	Lotus Notes Document Item Specification .....	715
5.91.	Lotus Notes Item Specification: General .....	716
5.92.	Lotus Notes Settings dialog .....	718
5.93.	Managing service projects.....	719
5.94.	Name Generation Algorithms.....	721
5.95.	Name Generation: Default input names .....	723
5.96.	Name Generation: Embedded algorithms.....	724
5.97.	Name Generation: Formatting functions .....	724
5.98.	Name Generation: Iteration .....	725
5.99.	Name Generation: Manage algorithms .....	726
5.100.	Network bar - Count users.....	728
5.101.	Network data .....	729
5.102.	Open UMRA project.....	729
5.103.	Name Generation: Setup algorithm methods .....	729

---

5.104.	Password generation .....	731
5.105.	Script action property value .....	732
5.106.	Scheduler .....	733
5.107.	Set items .....	735
5.108.	Setup scheduling.....	735
5.109.	Setup scheduling - Exceptions .....	739
5.110.	Setup scheduling - Adding an exception .....	745
5.111.	Setup scheduling - Preview.....	747
5.112.	Script action property value with yes/no option.....	748
5.113.	Script action property value output .....	748
5.114.	Script action property value - Output only.....	748
5.115.	Search and replace .....	748
5.116.	Security - Access Control Settings.....	750
5.117.	Security - Adding accounts and permissions .....	751
5.118.	Security - Detailed permissions settings.....	752
5.119.	Security - Overview.....	754
5.120.	Security - Owner .....	755
5.121.	Specify file input data .....	756
5.122.	Specify group names.....	757
5.123.	Specify input .....	757
5.124.	Specify input name .....	757
5.125.	Specify new name for UMRA project .....	758
5.126.	Specify radio button text info .....	758
5.127.	Specify variable info.....	758
5.128.	Task scheduler overview settings.....	759
5.129.	Task scheduler overview window.....	759



---

5.130.	UMRA Console - Command Line Options .....	763
5.131.	UMRA Project Component - File data.....	765
5.132.	UMRA Project Component - Form .....	765
5.133.	UMRA Project Component - Network data .....	765
5.134.	UMRA Project Component - Preview .....	766
5.135.	UMRA Project Component - Script .....	766
5.136.	UMRA Project Properties - Description .....	767
5.137.	UMRA Project Properties - Form Fonts .....	767
5.138.	UMRA Project Properties - Form options .....	767
5.139.	UMRA Project Properties - Format.....	768
5.140.	UMRA Project Properties - General.....	769
5.141.	UMRA workspace .....	770
5.142.	UMRA Project Properties - Initial variables .....	770
5.143.	UMRA Project Properties - Network data .....	771
5.144.	UMRA Project Properties - Options.....	771
5.145.	UMRA Project Properties - Security.....	771
5.146.	UMRA service - Advanced options .....	772
5.147.	UMRA service - license .....	773
5.148.	UMRA Service - service access.....	773
5.149.	UMRA service deletion - Delete all files .....	774
5.150.	UMRA service installation - Admin group.....	774
5.151.	UMRA service installation - Server .....	774
5.152.	UMRA service installation - Port.....	775
5.153.	UMRA service installation - Service account .....	775
5.154.	UMRA service installation - Service directory .....	776
5.155.	Variable generic table .....	776

5.156.	Variable list .....	778
<b>6.</b>	<b>No help available</b>	<b>778</b>
<b>7.</b>	<b>Index</b>	<b>779</b>

---

# 1. Welcome to UMRA

Welcome to UMRA.

## What is UMRA ?

UMRA, User Management Resource Administrator, is a software solution for the easy specification, execution and management of **network administration** related tasks.

Automated management of user accounts, (home) directories, shares, and e-mail are just a few examples of its use.

Its primary focus is on the management of Microsoft's Active Directory, and many other administrative network resources in networks based on Microsoft Windows.

It also has the capability to directly manage LDAP based directories, SQL databases, and IBM's Lotus Notes environments.

Administrators can use UMRA to create form-based applications that perform specific network tasks, and delegate the execution of those task to specific users in the network (for example helpdesk employees).

Additionally to tasks started by means of forms , tasks can also be started on a regular basis by means of the build-in scheduling mechanism, or started by an external application (for instance the company web site) using either the available command-line application, or by means of the UMRA COM interface.

It is also possible to let UMRA cooperate with many Human Resource systems, for example to automatically update the appropriate network resources when users are added to the HR system.

## The main modules

### 1) The UMRA console.

This is the main application installed by the setup program of UMRA. It is used by an administrator to create and configure the UMRA Project(s) that build the task. Each project contains a script with one or more script actions.

Projects can be created locally in the console, and be executed by the console.

The console is also used to install and configure the UMRA service, and to create and configure form based, scheduled and automation projects on the server.

### 2) The UMRA service

The UMRA service maintains and executes UMRA projects. All projects (tasks) that require forms, delegation, scheduling, or are started by external programs, are maintained by the service.

### 3) The UMRA Delegation Client

The UMRA Delegation Client, is a client application that is run by the end user (eg. the helpdesk employee) to access and fill out the forms on the UMRA Service in order to perform a specific administration task.

More information about the basic working of UMRA can be found in the *UMRA User Guide* on page 1

## 2. Release notes

Release notes

### 2.1. User Management Resource Administrator release notes

Version 10.9 Build 1664 may 18 2012

#### Major changes

1. **Action: Get Active Directory permission (Exchange 2010)** (new). Added support for the "Get-ADPermissions" cmdlet in Exchange (1661, 25/april/2012)
2. **Action: Manage Active Directory Permissions (Exchange2010)** (new). Allows to set/remove access rights on AD objects in the exchange environment. Mainly Used to Implement the "Send as" rights on a mailbox, as this is not supported by the mailbox permissions actions.(1661, 26/april/2012)
3. **Action: Manage Recipient Permissions(Exchange2010)** (new). Allows to set/remove "SendAs" rights in a hosted exchange configuration (1661, 27/april/2012)
4. **PSM:** (enhancement). The Password Synchronization Manager software plugin on the DC can now optionally automatically switch between two or more UMRA services in case of connection failures. In such a configuration, when the primary UMRA service is taken offline, synchronization notifications are automatically rerouted to an alternative UMRA service so no notifications are missed.

#### Fixes and enhancements

1. **Action: It's Learning Get person info** (enhancement). Added support to read the "Children" of a user. (1660, 02/apr/2012)

2. **Action: It's Learning Add child to parent** (enhancement). Added. (1660, 02/apr/2012)
3. **Action: It's Learning Create Person** (enhancement). Added support to create a parent. (1660, 02/apr/2012)
4. **Action: It's Learning Get persons** (enhancement). Added a column Children, containing the children of the user. (1660, 02/apr/2012)
5. **Atvo3 (SOM)** (enhancement): Added support for gzip compression. (02/apr/2012)
6. **Action: SAP Set user detail** (new): The parameter name is now always converted to uppercase. (1660, 02/april/2012)
7. **Action: Exchange2010 Create** (enable) mailbox. (fix) Parameter "ManagedFolderMailboxPolicyAllowed" is now only passed on when set to "true" (1660,10/april/2012)
8. **Action: Afas get employees** (fix): The parameter Active Reference Date works now as expected. (02/apr/2012)
9. **Action: Create user and mailbox** (Exchange 2010) (enhancement). Added support for AddressListPolicy . (1661, 16 april 2012)
10. **Action: Edit Mailbox (Exchange2010)** (enhancement). Added support for AddressListPolicy. (1661,/16/april 2012)
11. **Action: Format variable value (enhancement)**: The function "Remove diacritical marks" wil now also convert unicode characters 0x1E60, 0x1E62, 0x1E64, 0x1E66, 0x1E68 to 'S' and 0x1E61, 0x1E63 0x1E65, 0x1E67,0x1E67, to 's' . (1661, 28/april/2012)
12. **Action: Create user (AD)** (fix). Extended the support for creating accounts in Organizational Units, in case the name of the OU contains characters that are illegal in a LDAP string. When the Organizational Unit is specified as a separate property, illegal characters are now automatically escaped correctly. Also implemented in the "Get user (AD)", "Create contact (AD)", and "Create object (AD)" actions. (1661, 01/may/2012)
13. **Action: Generate Generic table** (fix). Fixed a small memory leak when using database queries. (1661, 02/may/2012)
14. **Action: Manage table data - Remove Duplicate rows (case insensitive)** (new): A new action option is added to support case insensitive compare when removing duplicate rows (1661, 02/may/2012).

## Version 10.8 BUILD 1659 (BETA) feb 14 2012

### Fixes and enhancements

**UMRA COM** (fix/enhancement). When the IUmra method "ExecuteProjectScript". is called, it sends its list of client-side variables to the UMRA service, as part of the initialization of the script to be executed. After execution the modified list as generated by the project is returned to the client. Any next call to ExecuteProjectScript will use this modified list for initialization. When the datasize of this list in the client exceeded the limit of 5MB, the call would fail, and result in an critical error. This limit has been lifted.

**Version 10.8 Build 1658, Dec 23 2011****Major changes**

1. **Office 365 Connector (new):** The Office 365 environment is supported with a set of dedicated UMRA actions. (1654, 06/December/2011)

**Fixes and enhancements**

1. **Action: New MoveRequest (Exchange 2010)** (enhancement). Added support for the "AcceptLargeDataLoss" parameter.(1656, 12/dec/2011)
2. **Action: Edit Mailbox (Exchange 2010)** (enhancement). Added support for the "RoleAssignmentPolicy" parameter.(1656, 12/dec/2011)
3. **Action: List MoveRequests (Exchange 2010)** (fix). The 'RequestStyle' property is now returned in the table instead of the obsolete 'MoveType' property.(1656, 12/dec/2011)
4. **Action: Load Ldap modification data** (enhancement). The resulting LDAP structure is no longer unconditionally logged in the Umra log files. This logging is now optional, and is off by default (1654,30/nov/2011).
5. **RPC Communication Layer (Fix):** Fixed an issue in the use of the memory manager for the RPC layer, as for example used by the UMRA Com object. This could cause the calling application to crash if a UMRA com method "execute project" was executed concurrently with another RPC using the same memory manager .This has been fixed. It most notably solves an issue for the SSRPM service when executing UMRA projects on password resets. (1656, 12/dec/2011).
6. **Action: Copy directory** (fix): When the input for this action is invalid such that the specified destination directory itself is within the source scope, an endless recursion can occur. Basic validity checks to prevent this have been added. (1650,12/october/2011)
7. **Action: Send HTML mail message** (fix): Unicode Characters in the mail body text that are not in the standard ascii set, are now encoded as e.g. &#180. This ensures that the mail client program can display them properly. (1650,10/october/2011)
8. **Action: Google Get users (advanced)** (fix): When retrieving the users, the column names from column 8 are incorrect. The name of column 9 was overwritten with the name of number 10. Number 10 with the name of 11 etc. Therefor only 63 column names where shown. The contents of the table where correct. Now the column names are correct. If a join is used on columnname of one of the columns above number 8 make sure the join is working as intended. (1650,5/oktober/2011)

9. **Action: Manage table (fix):** When creating a new table, or adding a new column, now each column will automatically get a default column name. This is because some other operations may depend in specific circumstances on the availability of a column name. (1650,4/october/2011)
10. **Action: Format variable value (enhancement):** The function "Remove diacritical marks" will now also convert unicode characters 0x1E20 en 0x1E21 to 'G' and 'g' respectively. (1650,28/september/2011)
11. **Service Installation (Enhancement):** Increased minimum delay threshold in any query for the servicesatus. This should prevent initial connection issues after upgrade of UMRA service (1650, 29/september/2011)
12. **RPC Communication Layer (Enhancement):** Communication between different UMRA components has been modified to support ip6 networks.(1649,28/september/2011)
13. **Action: Convert to multi value variable (fix):** Some of the memory resources used in this action were not correctly released. This could cause a growth of the memory used by the Umra service when this action was used frequently. This has been fixed. (1648,22/september/2011)
14. **TeleTOP (enhancement):** Added a option to set the maximum number of worker threads.(1648,22/september/2011)
15. **N@TSchool (fix):** Groups can now successfully be retrieved recursively.(1648,22/september/2011)
16. **IT's Learning (enhancement):** An update to the IT's Learning Software has enabled us to implement significant performance enhancements.(1648,22/september/2011)
17. **IT's Learning (fix):** Solved a memory leak in the connector.(1648,22/september/2011)
18. **Action: SAP Generic function modules (fix):** : Fixed the output of single numeric and date fields and added support for single string fields.(1648,22/september/2011)

Version 10.7 Build 1648, June 30, 2011

#### Major changes

1. **Exchange 2010 (Enhancement).** Added several actions in order to support the export of mailboxes to .pst files. (requires Exchange SP1) (1647,23/june/2011)
2. **Exchange 2010 (Enhancement).** Extended the "Create user and mailbox" action to support the creation of linked mailboxes. (1647,23/june/2011)
3. **Exchange 2010 (Enhancement).** Extended the "create user and mailbox" action to support the creation of room and equipment mailboxes. (1647,23/june/2011)

## Fixes and enhancements

1. **PowerShell: Dynamic actions** (fix): When the powershell script building from the xml specification generates an exception in the UMRA service, further processing of the action is cancelled to prevent potential critical errors in the Umra service engine. (1640, 20/December/2010)
2. **Action: Get variable length** (new): New action added that calculates the number of characters in a text variable. (1640,27/December/2010)
3. **Action: Create user (AD)** (fix): When setting the "User cannot change password" flag in this action, there was an error that could lead to a crash when specific inheritable rights were pre-existing on the organizational unit in which the user was to be created. This has been fixed. The fix applies also to all other actions that set this flag. (1640,28/december/2010)
4. **Action: Set variable** (enhancement): The option "resolve variable names in value immediately." is now also implemented for text-list variables. (1641,29/December/2010)
5. **Action: Send Mail Message** (fix): The Send Mail Message action would truncate the mail message send if there were any genuine UNICODE characters in the text (character codes higher than 255). This is fixed. Note that the resulting text message is encoded with code page 1252. (1641,30/December/2010)
6. **General** (fix): General issue with string conversion to and from unicode has been fixed, that could cause truncation of strings with non-standard characters. (1641, 30/january/2011).
7. **UMRA console** (fix): When opening multiple projects at once, all projects that are not already open in another workspace, will now open in the originally active workspace. Previously the default target workspace could change if a project was encountered that was already open in another workspace. (1641,30/march/2011)
8. **Action: Format variable** (enhancement). The variable to be formatted can be specified indirectly e.g. "%name%" where the variable %name% contains the name of the actual variable to be formatted, (1641,31/march/2011)
9. **Action: If then Else** (fix): The evaluation of the equations of the types like "older than # days" would cause an exception for the date 1/1/1601 (a.k.a. 'never') when used in countries with negative time zones. This is fixed. (1641, 5/april/2011)
10. **Action: If then Else** (enhancement): The evaluation of the type "older than # days" would not differentiate between dates before 1 jan 1970. Now it differentiates between dates from 2.jan 1601 upwards. (1641, 5/april/2011)
11. **Action: Setup LDAP session** (fix): The "on error" handler was not always called if an error occurred. This has been fixed. (1641, 5/april/2011)
12. **Google Apps connector** (fix): The "change password at next logon" option, that can be specified when creating or editing Google users, was not effective. This has been fixed (1641, 8/april/2011).
13. **Action: Lotus Notes Set item(s)** (fix): Setting an item of type "textlist" to an empty list could cause a critical exception. This has been fixed. (1641, 13/april/2011)



14. **Action: Lotes Notes Set Item(s)** (fix): Items of type "textlist" where not completely stored when exporting the project to an xml file, resulting in an empty list upon import. This has been fixed. (1641, 13/april/2011)
15. **Action: Manage multi-text value variable** (fix): The option "sort values in descending order" would clear the values. This has been fixed. (14/april/2011)
16. **Action: List files and/or directories** (change): In more circumstances the data collection will continue after an error has been encountered. (18/april/2011)
17. **Action: SAP Generic function modules** (new): A new action is added to support multiple custom RFC/BAPI function modules to be executed within the same SAP server session. (1641, 18/april/2011)
18. **Action: Manage Table Data** (fix): When Using de Table Data Operation "Complete rows", the resulting column count is now updated to reflect the size of the largest row. (1641,22/april/2011)
19. **Action: Edit user (no AD)** (fix): The account expiration date can now successfully be specified by means of a variable. (1641,22/april/2011)
20. **Action: Create user (no AD)** (fix): The account expiration date can now successfully be specified by means of a variable. (1641,22/april/2011)
21. **UMRA COM** (fix): When the IUmra method "ExecuteProjectScript". was called without a prior successful call to the method "Connect", this could cause an exception. This has been fixed. (1642,06/may/2011)
22. **UMRA COM** (Enhancement): The UMRA com object has been extended with two methods in order to allow access to individual records of the resulting log of the ExecuteProjectScript method. (1642 10/may/2011)
23. **UMRA COM** (Enhancement): The UMRA com object has been extended with a method "GetVariableInfo". to retrieve information of the type of data contained in a variable. Main purpose is to be able to determine whether or not the variable contains a table. (1643, 19/May/2011)

Version 10.6, Build 1640, December 17, 2010

#### Major changes

1. **AFAS Profit connector** (new): The HR-system AFAS is supported with a set of dedicated UMRA actions. (1638, 3/December/2010)

#### Fixes and enhancements

1. **Google** (fix): For all Google script actions, when an action returns result data as a table, and the action succeeds but the result set is empty, a table with 0 rows is generated. Previously no output was generated at all for the table variable when the result set was empty. Now table formatting functions will therefore work correctly on the result variable also if the result set is empty.(1635, 03/November/2010)

2. **Google (fix):** Extended the log information in case Google returns errors as text/plain instead of html or xml. (1637, 18/November/2010)
3. **Google (fix):** The UMRA service will stop all Google related activity when it receives a stop command. (1637, 22/November/2010)
4. **Action: Google get group info (fix):** Fixed a crash when this action was used on a closed connection. (1637, 18/November/2010)
5. **Action: Google get user info (advanced) (fix):** Fixed a rare situation causing a table row mismatch resulting in a empty table. (1637, 18/November/2010)
6. **Action: Google get user info (advanced) (fix):** Advanced user information is now returned correctly (especially birthdate and organizations). (1637, 18/November/2010)
7. **Action: Google rename user (fix):** Fixed some problems when renaming and using the cache. (1635, 05/November/2010)
8. **Action: Google add nickname (fix):** Fixed the problem of adding a nickname to a user when the user name was not specified in lower case. (1635, 05/November/2010)
9. **PSM: Installation (fix):** When the installation of the PSM notification package had been initiated from a UMRA console running on a Windows XP 32 bit OS, the PSM dll would fail to load on 64 bit domain controllers. This has been fixed. (1637, 05/November/2010)
10. **Action: Create User (AD) (fix):** If the action failed because of rejection of the password by windows, the error was logged, but the action returned a success code. Now it correctly raises an error.(1736, 23/November/2010)
11. **Action: Create user and mailbox(exchange 2007) (fix):** The specified "display name" property is no longer ignored by the action. (1637, 23/November/2010)
12. **License Check (fix):** The license check failed for licenses issued to an Organizational Unit that had special characters in its name (like "+students"). This is fixed (1637,26 nov 2010)
13. **Action: Update Group Memberships (AD) (fix):** If the user to change was located in an Organizational Unit with a forward slash ('/') in the name, the action failed. This is fixed. (1637, 26 November 2010)
14. **Action: Format variable value (enhancement):** The restriction on the maximum supported variable size has been relaxed by a factor 1000. It will now only log a warning, except for extreme cases. (1637,30 November 2010)
15. **UMRA Forms table (adjustment):** When specifying the column layout of a forms table at design time, the column widths are not anymore automatically resized to a total of 100% of the form width; instead, if the total width exceeds 100%, a horizontal scroll bar is shown in the resulting form. (1637,1 December 2010)
16. **Umra Forms table (adjustment) :** When specifying the column layout of a forms table, double clicking on an already configured column does not anymore remove the column from the list. (1638, 1 Dec 2010)

17. **Action: Set attribute (AD)** (fix): When the AD object variable to modify was specified, but contained no valid object, an error was logged and the action was not executed, but the "on error" actions were not performed. Now they are. (1638, 2 Dec 2010)
18. **Action: Send HTML mail message** (fix): Mail messages with text or .csv attachments did not display properly in some specific email clients. This has been fixed.
19. **Action: Send HTML mail message** (fix): Sending an email message could create and leave a temporary file in a root directory of the computer. Now it does not.

### Version 10.5, Build 1634, October 29, 2010

#### Fixes and enhancements

1. **FIPS compliancy** (fix): The UMRA Powershell Agent service is updated to use FIPS compliant encryption algorithms only (1632, 7/September/2010)
2. **Action: Setup Exchange Session (Exchange 2010)**. (fix) Automatic reconnect on timeout works now also without explicitly specified credentials. (1633, 10/October/2010)
3. **SAP** (fix): Support for the user detail 'PARAMETER1' in the Set detail and copy user actions. (1633, 23/September/2010)
4. **Action: Send HTML mail message** (enhanced): The *port number of the SMTP E-mail server* can now be specified. (1631, 2/September/2010)
5. **Action: Google Create contact** (fix): Fixed the crash, caused by using this action (1630, 20/August/2010)
6. **Action: Google Create user** (fix): Fixed the problem when suspended users were created in the cache, they also were connected to an invalid profile. (1630, 20/August/2010)
7. **Action: Google Create user** (fix): Fixed the problem of users where allowed to be created, while a nickname with the same name as the new username, already existed. (1630, 20/August/2010)
8. **Action: Google Rename user** (fix): Fixed the problem of users where allowed to be renamed to a username which was already being used as nickname. (1630, 20/August/2010)
9. **Action: Manage mailbox email addresses** (Exchange2007) (fix): The Remove address option is now also applied to existing projects. (1630, 23/August/2010)
10. **Action: Google Get contact info** (new): A new action is added to retrieve all the single valued attributes and all the primary table entries from a contact. (1630, 24/August/2010)
11. **Action: Google Contact remove** (fix): Fixed an error, which prevented the contact from being removed, when the contacts cache was not used. (1630, 24/August/2010)
12. **Action: Google get user info** (enhancement): Added the nicknames as output property. (1631, 01/September/2010)
13. **Action: Google get user info (advanced)** (enhancement): Added the nicknames as output property. (1631, 01/September/2010)

14. **Action: Google Remove member from group (fix):** The action will now respect the 'Remove as owner' property. (1633, 05/October/2010)
15. **Google (fix):** UMRA could crash if one script was closing the connection more times as it opened connections and an other script was at the same time processing modifications. (1630, 20/August/2010)
16. **Google (fix):** Fixed a rare condition where a Google Close Connection could close the connection of an other script. (1630, 24/August/2010)
17. **Google (enhancement):** Added more error checking and more information in case of errors in requests. (1630, 24/August/2010)
18. **Google (fix):** Fixed the problem of not reinitializing the cache if the process changes got errors other than 1300 or 1301 (entityexists or entitydoesnotexist). (1630, 20/August/2010)
19. **Google (fix):** Support for all unicode 2 bytes characters including diacritical marks and XML special characters like '<'. (1631, 31/August/2010)
20. **Google (fix):** Better support when retrieving big environments and the user/profile data is changed while retrieving. (1632, 06/September/2010)
21. **Action: @VO Get students courses/classes (new):** A new action to retrieve a students courses and classes from an @VO3 environment. (1631, 01/September/2010)
22. **Action: @VO Get students custom fields (new):** A new action to retrieve a employees custom fields from an @VO3 environment. (1631, 01/September/2010)
23. **Action: @VO Get employees custom fields (new):** A new action to retrieve a employees custom fields from an @VO3 environment. (1631, 01/September/2010)
24. **Action: @VO Get employees courses/classes (new):** A new action to retrieve a employees courses and classes from an @VO3 environment. (1631, 01/September/2010)
25. **Action: @VO3 Get caregivers (fix):** The action will return all the care givers, instead of just the ones, who have only one student when the normal filters or the legal representative filters are used. (1633, 10/September/2010)
26. **@VO (fix):** Fixed some school year and password issues causing the use of the current school year and corrupting the passwords in the authorization functionality. (1631, 01/September/2010)
27. **TeleTOP (fix):** The course year is not send to TeleTOP when an old environment is used. (1631, 01/September/2010)
28. **Action: N@TSchool Get user info (enhancement):** The action has an option to not return the user attributes to improve speed (1633, 7/September/2010)
29. **Action: N@TSchool Get user info (fix):** Correctly return the user attributes, when requested. (1633, 7/September/2010)
30. **N@TSchool (fix):** Now all traffic is routed thru a proxy when requested. Some traffic was not routed thru a proxy, when proxy information was specified. (1633, 7/September/2010)

31. **UMRA COM (fix):** The UMRA client session between the UMRA COM object and the UMRA Service is automatically unique for each instance created for the UMRA COM object. In previous version this could cause a problem when the UMRA COM object was accessed by the same user in multiple ASP or ASPX pages simultaneously (1631, 7/September/2010).

## Version 10.5, Build 1630, August 20, 2010

### Fixes and enhancements

1. **Action: Send HTML mail message (new):** A new action to send HTML E-mail message that can also contain one or more attachments. See *Script Action: Send HTML mail message* for more information. (1626,10/August/2010)
2. **Action: SAP Generic function module (new):** A new action is added to support any RFC/BAPI function module. See *SAP - SAP Generic function module* for more information. (1625, 9/August/2010)
3. **Action: Setup Exchange Session (Exchange 2010).** (fix) Modified optional prerequisite test to better reflect the current prerequisites. (1625, 27/July/2010)
4. **Action: Edit Distribution Group (Exchange 2010)** (fix) The "Room List" switch is now not specified at all (instead as false) when not explicitly set, to prevent errors in outlook live, that does not support this parameter. (1625,27/July/2010)
5. **Action: Add Distribution Group member (Exchange 2010)** (new): A new action is added to add an Exchange 2010 distribution group member to an existing distribution list. See *Introduction to Exchange 2010* (see "Introduction Exchange 2010" on page 1) for generic information on using Exchange 2010 UMRA actions. (1625, 27/July/2010)
6. **Action: List Distribution Group members (Exchange 2010)** (new): A new action is added to list the members of an Exchange 2010 distribution list. (1625, 27/July/2010)
7. **Action: Remove Distribution Group member (Exchange 2010)** (new): A new action is added to remove a member from an Exchange 2010 distribution list .(1625, 27/July/2010)
8. **Action: Sap get users** (fix): When the search pattern is was used, it was ignored by the action. This issue is fixed (1623, 16/June/2010)
9. **Action: List mailboxes (Exchange 2010)** (enhancement): Added MailboxPlan column to advanced table. so it can be listed if it is supported in the used Exchange 2010 environment, for instance in Outlook Live. (1623,17/June/2010)
10. **Action: Create user and Mailbox (Exchange 2010)** (enhancement): Added optional MailboxPlan specification for Outlook Live support (1623,17/june/2010)
11. **Action: Edit Mailbox (Exchange 2010)** (enhancement): Added option MailboxPlan specification for Outlook Live support. (1623,17/june/2010)
12. **Action: List mailboxes (Exchange 2010)** (enhancement): Added optional RecipientTypeDetails filter parameter. (1623,17/june/2010)

13. **Action: List Users (Exchange 2010)** (enhancement) Added optional RecipientTypeDetails filter parameter. (1623,17/june/2010)
14. **Action: List Contacts (Exchange 2010)** (enhancement) Added "Error when not found" option flag. (1623,17/june/2010)
15. **Action: Sap Set user detail** (fix): When the detail value is table with multiple rows, not only the last row is send to SAP multiple times, but all different rows are send to SAP. (1623,07/July/2010)
16. **Action: Manage mailbox email addresses** (Exchange2007) (fix): The Remove address option worked incorrectly (1623, 15/july/2010)
17. **Action: Create Mail User (Exchange2010)** (fix): Removed the mandatory tag from some action properties to allow use for Outlook Live. (1627,13/august/2010)
18. **Action: Create Mail Contact (Exchange2010)** (fix): Removed the mandatory tag from some action properties to allow use for Outlook Live. (1627,13/august/2010)
19. **Action: Sap Remove user detail** (fix): Sometimes the detail was not cleared in rare conditions, this is fixed. (1626,12/August/2010)
20. **Google** (enhanced): The support of google apps is enhanced. See *UMRA Google module* for more information. (1623, 16/June/2010)

#### **Version 10.4, Build 1623, May 28, 2010**

##### **Major changes**

1. **SAP** (enhanced): The support of SAP systems is enhanced. See *UMRA SAP module* for more information. (1613, 15/April/2010)
2. **Exchange 2010** (New): Official support for Exchange 2010 has been added. Over 50 dedicated actions to support various exchange 2010 features to manage mailboxes, distribution lists, public folders and more (1619, 18/may/2010)
3. **@VO 3 connector** (new): The hosted student information system @VO is supported with a set of dedicated UMRA actions (1615, 22/April/2010)
4. **Avetica Moodle connector** (new): The hosted student information system Moodle is supported, when hosted by Avetica, with a set of dedicated UMRA actions (1615, 22/April/2010)
5. **Edictis connector** (new): The webservices of Edidictis are supported with a set of dedicated UMRA actions (1615, 22/April/2010)
6. **Google Apps connector** (new): The action **Google Rename user** is added to the Google Apps connector (1615, 28/April/2010).

##### **Fixes and enhancements**

1. **Action: Set variable** (enhancement): The name of the variable to create now may contain names of earlier defined variables. This allows for the names of the variable to be dynamic. For instance

%%NewVarName%%. or %Administrator of %CurDomain%%. T4E\_ID 680 (1601, November 27, 2009)

2. **Action: Manage Table data** (enhancement): Table data operation "Get the number of table rows", now sets the found rowcount to 0 even if the specified table does not exists. Previously the rowcount variable was not generated at all in that situation. This could cause issues if the script did not explicitly check for errors. t4E\_ID 777 (1601, November 26, 2009).
3. **Action: Delete Directory** (fix): When the "Delete directory option" was specified in order additionally delete the specified directory itself, and an error occurred in the deletion process of this directory, the "on error" handler was not invoked. Now the "on Error" handler will be correctly invoked. T4E\_ID 776 (1601, November 26, 2009).
4. **Action: If-Then-Else** (fix): When specifying a date-time value as a condition criteria, the value is updated when the interface dialogs are re-opened. The difference in time corresponds with the difference between local and GMT time zone settings. The issue is fixed (1615, 22/April/2010)
5. **Action: Update database** (enhancement): An option is added to prevent to contents of database statement being shown in the log files. This is useful if the statements contain sensitive information, for instance a password. See *Script Action: Update database - SQL Statements* on page 539 for more information. (1615, 28/April/2010).
6. **Action: Execute command line** (enhancement): A property is added to prevent the command line being shown in log file. This is useful is the command line contains sensitive information, for instance a password (1615, 28/April/2010).
7. **Action: Format variable value** (enhancement): A option is added to prevent the input and output text values from being shown in the log files (1615, 28/April/2010).
8. **UMRA Forms** (fix): When a UMRA Forms project used an initial project and the initial project accessed the UMRA Powershell Agent service, memory errors could cause the UMRA Service not to respond correctly. This issue is fixed (1613, 19/April/2010).
9. **UMRA Forms** (fix): When the UMRA Forms client is started, the File, Exit menu option did not always work when the application was not connected to an UMRA Service. The issue has been resolved (1613, 19/April/2010)
10. **UMRA Forms** (fix): When an UMRA Forms table was configured with multiple selection disabled and the index of the selected item was stored in a variable, the index could be incorrect if the end-user resorted the UMRA Forms table. The issue has been resolved (1614, 27/April/2010).
11. **UMRA Forms** (fix): When a generic table, generated in a previously executed project, is sorted before it is shown in the UMRA Forms client, the indices of selected items could be incorrect. This issue has been resolved (1615, 28/April/2010)
12. **NTFS file system** (fix): When specifying the file and directory security settings an error can occur when specifying a deny permission entry. UMRA will deny the permissions as specified, but also the so called synchronize permission is denied. The issue has been resolved (1614, 20/April/2010).

13. **UMRA Automation** (fix): When scheduled projects run for over 24 hours, the UMRA Session could expire in previous versions. This might cause problems when UMRA Session variables are used. The problem has been resolved (1615, 22/April/2010)
14. **UMRA COM** (enhancement): The UMRA COM object is extended with method **HideVariable** to prevent the contents of the variable data from being shown in log files. See *HideVariable* for more information (1619, 3/May/2010)
15. **Generic table** (fix): The octet string data type, used in Active Directory to represent for instance SID and GUID values is now supported in UMRA (1618, 29/April/2010).
16. **TeleTOP connector** (enhancement): Added support for the TeleTOP course code enhancements. Also improved connection stability in unstable network environments (1615, 22/April/2010)
17. **Google Apps connector** (fix): Fixed the retrieval of user memberships information when the cache is overridden (1615, 22/April/2010)
18. **Google Apps connector** (enhancement): Added support for renaming users (1615, 22/April/2010)
19. **It's Learning connector** (enhancement): Improved the performance and overall connection stability of the connector (1615, 22/April/2010)

### Version 10.3, Build 1601, November 19, 2009

#### Major changes

1. **TOPdesk connector** (new): The helpdesk information system *TOPdesk*, <http://www.> is supported with a set of dedicated UMRA actions (1587, 26/August/2009)
2. **Google Apps connector** (new): The information system Google Apps is supported with a set of dedicated UMRA actions (1587, 26/August/2009)
3. **TeleTOP connector** (new): The student information system *TeleTOP* is supported with a set of dedicated UMRA actions (1587, 26/August/2009)
4. **N@tSchool connector** (new): The student information system *N@tSchool* is supported with a set of dedicated UMRA actions (1575, 10/June/2009)
5. **It's Learning connector** (new): The student information system *It's Learning* is supported with a set of dedicated UMRA actions (1599, 4/November/2009)
6. **Aura connector** (new): The school library system *Aura* is supported with a set of dedicated UMRA actions (1575, 10/June/2009)
7. **Execute command line at UMRA Forms client** (new): When an UMRA form is submitted, as a response, a command line can be executed by the UMRA Forms client computer. See *Form action - Execute command line at client workstation* for more information (1560, 2/April/2009).
8. **Get Out-Of-Office info (Exchange 2007)** (new): The action collects the Out-Of-Office information of a particular account. See *Script Action: Get Out-Of-Office info (Exchange 2007)* for more information (1572, 2/June/2009)



9. **Set Out-Of-Office info (Exchange 2007) (new):** The action sets the Out-Of-Office information of a particular account. See *Script Action: Set Out-Of-Office info (Exchange 2007)* for more information (1572, 2/June/2009).

## Powershell

1. **Powershell Agent service - Session time to live (enhancement):** The time-to-live idle time of a *Powershell Agent service session* on page 78 can be configured by setting a registry value. See Registry settings for more information. (1570, 12/May/2009).
2. **Script action: Check Powershell Agent service session (new):** The action can be used to check if an previously created Powershell Agent service action is still available and removed upon expiration. See *Script Action: Check Powershell Agent service session* for more information (1570, 13/May/2009).
3. **Keep-alive signalling (enhancement):** To prevent expiration of idle Powershell Agent sessions, the UMRA Forms client sends keep-alive signal to the UMRA Service. The UMRA Service forwards these signals to the Powershell Agent service, keeping the sessions initialized through the UMRA Forms client alive. In case the UMRA Forms client is open for a longer period, e.g. a day, the session information is not lost.(1570, 19/May/2009)
4. **Variable list (enhancement):** The available variables that can be used in script properties, generated from dynamic actions, is now limited to the properties section of the dynamic actions. In previous version, also variable names from the script section were copied to the variable list. Since these variable are not available in the UMRA variable list, these variables should not be presented to the end-user. (1570, 22/May/2009)
5. **Powershell not installed (fix):** When Powershell is not installed on the machine that runs the Powershell Agent service, a correct error message is shown when executing the action **Setup Powershell Agent service session**. In previous versions, the Powershell Agent service and the call UMRA software could crash (1571, 25/May/2009)
6. **Powershell Agent service import library (fix):** When importing the Powershell Agent service library, some incorrect error message were shown. The issue has been resolved (1571, 25/May/2009)
7. **Project execution using UMRA COM (fix):** When using UMRA COM to execute projects that access the UMRA Powershell Agent service, a problem could occur causing projects not to terminate completely. The issue has been resolved. (1578, 17/June/2009)
8. **Powershell Agent service manual installation (enhancement):** A online help topic is added that described how to setup the Powershell Agent service manually. See *Manual installation of the Powershell Agent service* for more information. (1579, 30/June/2009)

## Lotus Notes

1. **Lotus Notes - action: Execute agent script (new):** The action creates, executes and deletes a Lotus script agent in an existing database. The action can be used for instance to automate the approval of administration process requests. See *Script Action: Execute agent script* on page 485 and *Lotus Notes example projects* (on page 54) for more information (1558, 30/March/2009).

2. **Lotus Notes - action: Execute agent script** (enhanced): The action is extended with some new properties to better control the action. See *Script Action: Execute agent script* on page 485 and *Lotus Notes example projects* (on page 54) for more information (1593, 28/September/2009).
3. **Lotus Notes - action: Get item size** (new): The action retrieves the size of a single specified Lotus Notes document item. See *Get item size* on page 464 for more information. (1558, 30/March/2009).
4. **Lotus Notes - action: Set item(s)** (enhanced): The action now checks the total size of the item. If the action would exceed the Lotus Notes item size limit, the action is not executed. The action size limit is 32k bytes for summary items and 64k bytes for all other items. (1558, 30/March/2009)
5. **Lotus Notes - action: Set item(s)** (enhanced): The action now checks the total size of the item. If the action would exceed the Lotus Notes item size limit, the action is not executed. The action size limit is 32k bytes for summary items and 64k bytes for all other items. (1558, 30/March/2009)
6. **Lotus Notes - action: Update profile document** (enhanced): The action now supports different Lotus Notes type values and can also be used to delete fields and/or sign profile documents only. See *Script Action: Update profile document* on page 478 for more information (1568, 24/April/2009).
7. **Lotus Notes - action: Update profile document** (enhanced): The action is further enhanced to add a profile document field that contains the date and time of the profile document signature. See *Script Action: Update profile document* on page 478 for more information (1569, 1/May/2009).
8. **Lotus Notes - action: Register person (advanced)** (fix): The action is extended with 2 new properties, 'Roaming - Replica servers' and 'Roaming - Create replica files in background' to support roaming profiles. See *Script Action: Register person (advanced)* for more information. (1585, 24/July/2009)
9. **Lotus Notes - example project 'Remove Roaming profile'** (new): An example project is added to show how to create an administration request to remove the roaming profile of a Lotus Notes user account. See *Lotus Notes example projects* (on page 54) for more information (1585, 24/July/2009).
10. **Lotus Notes - example project 'Lotus Notes ID Vault - Reset password'** (new): An example project is added show how to use the Lotus Notes ID Vault to reset password of user accounts in Lotus Notes. See *ID Vault* and *Lotus Notes example projects* (on page 54) for more information (1593, 28/September/2009).
11. **Lotus Notes - action: Delete document** (new): New properties are added to support another method to specify the note or document to be deleted. See *Script Action: Delete document* for more information (1593, 28/September/2009).

## Actions

1. **Update numeric variable - Convert number to text (format)** (new): The action is extended with the option to convert a number to a text value according to a C-language 'printf' format specification. See *Script Action: Update numeric variable* on page 550 for more information (1558, 30/March/2009).
2. **Execute command line** (enhanced): A property is added to specify the maximum output buffer size in case the output is to be stored in a variable. See *Script Action: Execute Command Line* on page 369 for more information (1579, 26/June/2009).

## Fixes and enhancements

1. **Action: Update date-time variable - subtract date-time value** (new): A new function is added to subtract a date-time value stored in a variable from another date-time value. See *Script Action: Update date-time variable* on page 552 for more information (1566, 17/April/2009).
2. **Action: Set variable** (fix): For hidden variables, in log files, the value of the variable was not shown but when the action was executed, it was shown. The issue has been resolved (1577, June 12, 2009).
3. **Set attribute (AD)** (fix): The escape sequences, introduced in UMRA build 1558, are changed to [\r], [\n], [\r\n] and [\t] to allow attribute specifications containing for instance \t: \\SERVERNAME\Share\tsmith. See *Script Action: Set attribute (AD)* on page 124 for more information. (1563, 9/April/2009).
4. **Add action to script - window update** (fix): When composing an UMRA script using menu option **Add action to script**, the script window is now updated correctly. In previous versions, the actions displayed were not always updated immediately. (1562, 7/April/2009)
5. **Script action: Set Terminal Services user settings** (fix): In special circumstances, UMRA could crash when the action was executed and failed with the following error message: Cannot determine NETBIOS domain controller name of domain controller... The issue has been resolved (1562, 7/April/2009).
6. **Powershell Agent service** (fix): When received a stop signal, the Powershell Agent service is now stopped more gracefully (1562, 8/April/2009).
7. **Generic table - LDAP table column name** (fix): The name of a column of a LDAP generic table is set to the name of the attribute. This was changed in build 1558 to the display name but causes problems in existing implementations. (1565, 15/April/2009)
8. **Vista - Windows Server 2008 - UAC** (fix): The UMRA Console is started with elevated administrative access on the Vista and Windows Server 2008 platforms. In previous versions, this was not the case. Depending on the system configuration access denied errors could occurs for instance when the UMRA Service was installed. (1566, 17/April/2009)
9. **Action: Update database** (fix): When testing the statements of action **Update database**, the option **to Run test on UMRA Service** is no longer available. In previous version, the option could be selected but was not functional (1566, 17/April/2009).
10. **64-bit UMRA COM DLL** (fix): In UMRA 10.1, build 1577, the 64-bit UMRA Automation DLL UmraCom64.dll had an incorrect version number (1577, June 12, 2009).
11. **PSM, UMRA session** (fix): When accessing an UMRA project through UMRA PSM, the global UMRA session list is now correctly updated. In previous versions, the UMRA Service was not updated correctly. (1588, August 31, 2009).

**Version 10.0, Build 1558, March 27, 2009****Major changes**

1. **Session support of the Powershell Agent service:** A Powershell Agent service session allows a more interactive usage of the Powershell runtime environment. For instance to store Powershell variables that can be used in subsequent Powershell scripts. For more information, see *Powershell Agent service session* on page 78. (1528, 10/November/2008)
2. **SAP support (new):** Over 30 actions are added to support SAP. The UMRA SAP actions can be used to create SAP accounts, reset passwords, add users roles and profiles and so on. See *UMRA and SAP* for more information. (1480, 5/September/2008)
3. **UMRA Console: Open referenced project (new):** When configuration a project with the UMRA Console application, the menu option **Open referenced project** opens the associated project for For-Each and Execute script actions. (1546, 29/January/2009).

**Actions**

1. **Get Out-Of-Office info (Exchange 2000/2003) (new):** The action collects the Out-Of-Office information of a particular account. See *Script Action: Get Out-Of-Office info (Exchange 2000/2003)* on page 174 for more information (1539, 6/January/2009).
2. **Set Out-Of-Office info (Exchange 2000/2003) (new):** The action sets the Out-Of-Office information of a particular account. See *Script Action: Set Out-Of-Office info (Exchange 2000/2003)* on page 178 for more information (1539, 6/January/2009).
3. **Set variable - hidden variable (enhancement):** With action *Set variable* on page 544 it is possible to hide the value of the variable. In log files, the value is not shown. (1528, 6/November/2008)
4. **Edit share (enhancement):** The property 'Cache parameter' is added to support *share caching options* on page 364 (1543, 28/January/2009).
5. **Set attribute (AD) (enhancement):** The action supports carriage return, line-feed insertion in the Active Directory attribute. See *Script Action: Set attribute (AD)* on page 124 for more information. (1546, 30/January/2009).
6. **Delete multiple variables (new):** The action supports *deletion of multiple variables* from the project variable list with a single action. (1546, 6/February/2009)

**Table management**

1. **Manage table data - Get the number of table columns (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
2. **Manage table data - Copy row (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).

3. **Manage table data - Copy multiple rows (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
4. **Manage table data - Copy table (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
5. **Manage table data - Remove multiple rows (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
6. **Manage table data - Remove a specified column (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
7. **Manage table data - Sort on column name (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
8. **Manage table data - Convert multi-value variable to table (new):** The action accepts single value variables. (1546, 11/February/2009).
9. **Manage table data - Replace column name (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
10. **Manage table data - Get column name (new):** The new action is added. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).
11. **Manage table data - Search table (new):** The action is extended with search features. See *Script Action: Manage table data* on page 528 for more information (1546, 5/February/2009).

#### **Powershell - dynamic actions**

1. **Time Powershell Agent service (fix):** The time as shown in log messages generated by the Powershell Agent service is now correct (1520, 21/October/2008).
2. **Upgrade of dynamic actions (fix):** The upgrade procedure of dynamic actions is enhanced (1520, 17/October/2008)
3. **Directory of Powershell Agent service (enhancement):** It is now possible to specify the directory where the Powershell Agent service is installed (1542, 13/January/2009).
4. **Powershell Agent service - UMRA Service (fix):** When multiple ( > 10) scheduled tasks access the Powershell Agent service simultaneously from within the UMRA Service, the RPC service can become to busy, causing errors and Powershell scripts not being executed. The error is now handled correctly and the RPC call is retried until it succeeds or the expiration period is passed (1506, 7/October/2008).
5. **Powershell Agent service - UMRA Service (fix):** In rare circumstances, the UMRA Service could crash when multiple scheduled tasks access the Powershell Agent service simultaneously. This is caused by some XML libraries not being thread-safe. The issue has been resolved (1506, 7/October/2008).

### UMRA COM object

1. **New interface methods (new):** A number of interface methods are added, mainly dealing with UMRA COM object tables. Methods are added to check license information, store tables in the variable list, manage table contents and column names using the UmraDataTable interface. See *UMRA COM object reference* on page 6 for more information (1543, 27/January/2009).

### Lotus Notes

1. **Lotus Notes - Update profile document (enhancement):** The action is extended: item flags can be specified for the updated profile document and the profile document can be signed when changes are applied (1536, 12/December/2008).
2. **Lotus Notes - Copy document (new):** Copy a Lotus Notes document from one database to another Lotus Notes database. See *Script Action: Copy document* on page 459 for more information (1536, 23/December/2008).
3. **Lotus Notes - Get quota (new):** Retrieve the quota and size of a Lotus Notes database. See *Script Action: Get quota* on page 445 for more information.
4. **Lotus Notes - Update profile document, 'Log archiving' (new):** Online help is updated to show how the action is used to set *log archiving for a Lotus Notes database* on page 478 (1543, 28/January/2009).

### Fixes and enhancements

1. **Password Synchronization Manager (fix):** Some memory issues are resolved in the Password Synchronization Memory DLL (1480, 10/September/2008)
2. **Password Synchronization Manager (fix):** An issue is fixed for domain controllers with a name of 15 characters. In previous versions, error 111 could occur, generated by the Password Synchronization DLL, running on the domain controller. (1506, 7/October/2008)
3. **XML (fix):** The handling of special characters (white-space, carriage return, line feed, tab) in XML export and import procedures is now correct (1480, 5/September/2008).
4. **XML (fix):** For some UMRA objects, the XML export was not complete, e.g. the result file did not contain all of the UMRA object data. This could lead to incomplete backups. (1480, 5/September/2008)
5. **XML (fix):** The indentation of exported projects to XML files is now correct. (1527, 30/October/2009)
6. **XML (fix):** When importing a project from an XML-file that contains a generic table with an imported file, the separator character settings are now correct. In previous version, the comma (,) separator was always selected (1546, 5 February/2009).
7. **Recent projects (fix):** In rare circumstances, the UMRA Console application lost the list with recent projects. The issue has been resolved (1515, 16/October/2008)

8. **Logging of service projects cache parameters** (enhancement) The projects cache parameters are now logged in the UMRA Service log in startup. The log message has the following format: Service projects cache initialized with parameters 'Enabled=1', 'Delay=300' (1522, 22/October/2008)
9. **Manage service projects** (fix): When the buttons of the 'Manage service projects' are clicked in a certain order, the UMRA Console application could crash. The issue has been resolved. (1528, 7/November/2009).
10. **Thread mechanism** (enhancement): To prevent delays in large networks, specific tasks are performed in separate threads. (1543, 14/January/2009).
11. **Form security - group selection** (enhancement): When setting the accounts for form project security, available groups are now by default shown in the dialog to select User and/or Groups (1542, 12/January/2009).
12. **Importing project files with period (.) in file name** (fix): It is now possible to import a project file with multiple periods (.) in the file name. Example: form.with.period.ufp (1543, 13/January/2009).
13. **Name generation - output variable** (enhancement): The names of output variables can now be renamed when configuration name generation algorithms. (1546, 29/January/2009).

## **Version 9.1 Build 1478, August 1, 2008**

### **Major changes**

1. **XML project file format** (new): UMRA supports the XML standard to import and export UMRA projects and scripts. See *UMRA XML project and script files* for more information (1458, 3/June/2008).
2. **Password Synchronization Manager (PSM)** (new): This new PSM module will catch every password change in a Windows Active Directory domain and start an UMRA project. The UMRA project will forward the password change to other systems and applications. Refer to the section on *PSM* on page 119 for more information (1474, 8/July/2008).

### **Groups**

1. **Update group memberships (AD)** (new action): The new action allows the addition, removal and synchronization of group memberships for an account. Lists can be specified for each operation. For example, when synchronizing, the user account (or other type of account) will only be a member of the specified groups when the action is completed. See [*Script Action: Update group memberships (AD)* on page 131] for more information (1437, 2/April/2008).
2. **Create group (AD)** (fix): The **Common Name** of a new group can now start with a #-character (1462, 12/June/2008).

## Tables

1. **Manage table data, Set column name** (enhancement): The action Manage table data now supports the function to set the name of a column (1433).
2. **Manage table data, Search cells with matching contents** (enhancement): In a table, search through all rows and the specified or all columns to find tables cells with matching text contents (1441, 14/April/2008).
3. **Manage table data, Complete rows** (enhancement): Search through all rows of a table and add empty text values to each row so that the total number of columns is equal for all rows. If all rows already have an equal number of cells, no changes are made (1441, 14/April/2008).
4. **Get user (AD)** (enhancement): The action is extended with property Globally Unique Identifier (GUID). When the user object is successfully retrieved, the GUID of the user account can be stored in a variable (1458, 2/June/2008).
5. **Get users table (locked out/Disabled/Password) (AD)** (fix): When the domain **Account lockout duration** is specified as 0 (account is locked out until administrator unlocks it), the action functions correctly (1462, 12/June/2008).
6. **Store indices of selected rows in table variable** (new): The indices of the selected rows of UMRA form tables, can be stored in a table variable (11/June/2008, 1462).

## Actions

1. **For Each** (enhanced): The dialog window to configure the For-Each action is extended with more options to configure the column variables passed to child project (1435, 21/March/2008).
2. **Execute command line** (enhanced): A property is added to allow the removal of carriage-return line-feed characters at the end of the output variable value (1438, 3/April/2008).
3. **Generate random number** (enhanced): For the minimum and maximum values, variable names can now be specified. (1441, 14/April/2008)
4. **Delete directory** (enhanced): The logic to calculate the correct directory name is improved to support directory names with multiple dots (.) in the full path (1441, 14/April/2008).
5. **Execute script** (enhanced): The action description as shown in the script window of an UMRA project now shows the name of the script to execute (1442, 18/April/2008).
6. **Get user (AD)** (fix): When an output variable is specified for the display name, and no display name is found for the user account, no UMRA error is generate. (1456, 29/May/2008).
7. **Format variable value** (new): The functions of the action 'Format variable value' can now be specified using variables. Also, when formatting functions are used in name generation algorithms, variables are supported (1437, 28/March/2008).
8. **Format variable value** (enhanced): The case conversion functions of the action now supports the special characters ä, Ä, ö, Ö, ü and Ü. (1471, 2/July/2008).



9. **Get attribute (AD) (fix):** When an attribute is not found, and property 'Convert to text flag' is set to 'No' and no errors must be generated if not found, the action now no longer generates an error (1462, 16/June/2008).
10. **Create local group (fix):** When the group cannot be created since it already exists, no error is generated if 'Error if group already exists' is set to 'No'. (1473, 9/July/2008).

## Exchange

1. **Manage mailbox e-mail addresses (Exchange 2007) (enhancement):** The action now contains an additional property **Domain controller** to allow specific server binding and avoid replication issues (1462, 16/June/2008).
2. **Enable distribution group (Exchange 2007) (new):** A new action to mail-enable distribution groups (1462, 16/June/2008).
3. **Set client access attributes (Exchange 2007) (new):** A new action to set client access attributes including Outlook Web Access (OWA), MAP, IMAP and POP (1462, 16/June/2008).

## Powershell

1. **Powershell agent service:** The service now supports more data types used to return output tables. For example the data of file system ACL's. (1425, 11/Feb/2008)
2. **Powershell agent service (fix):** A memory issue with encrypted data has been resolved. When using encrypted script phrases, the agent service could consume little memory resources that were not released properly. Eventually (after months or years without a reboot), this might cause problems for the computer running the Powershell Agent service (1425, 18/Feb/2008)
3. **Powershell return data (new):** Simple string and data values can be returned to UMRA with a more simple method. See *Single value output data* for more information (1467, 26/June/2008).
4. **Powershell encrypted variable input (new):** For input text properties, the value can be encrypted. In this case, the actual contents of the property is not shown in log files, UMRA script files and so on. See *Encrypted properties* for more information (1467, 27/June/2008).
5. **Powershell - Active Directory permissions management (new):** New UMRA actions are added to manage Active Directory permissions: *Script Action: Get AD permissions* on page 578, *Script Action: Add AD permission* on page 583, *Script Action: Remove AD permission* on page 588, *Script Action: Set AD permissions (advanced)* on page 593, *Script Action: Get owner* on page 595, *Script Action: Set owner* on page 597 (1473, 8/July/2008).
6. **Powershell - Group management (new):** New UMRA actions are added to manage Active Directory groups: *Script Action: Set Managed By* on page 599, *Script Action: Get (nested) group memberships* on page 601 (1473, 8/July/2008).
7. **Powershell - File system (new):** A new action is added to get disk space information: *Script Action: Get disk space* on page 602 (1473, 8/July/2008).

8. **Powershell - Active Directory utility (new):** A new actions is added to determine the role of the primary domain controller: *Script Action: Get PDC (AD)* on page 605 (1473, 8/July/2008).-

## Lotus Notes

1. **Lotus Notes - Support new action to reset the password of an Lotus Notes ID file.** See *Lotus Notes action: Generate recovery password* (see "Script Action: Generate recovery password" on page 440) for more information (1433).
2. **Lotus Notes - Support new action to configure Out-Of-Office.** See *Lotus Notes action: Out-Of-Office* for more information (1433).
3. **Lotus Notes - Move person (advanced) (new):** The existing action **Move person** cannot be used to move a person if the person is currently located in an organizational unit. To support this operation, the action **Move person (advanced)** is added. See [*Script Action: Move person (advanced)*] on page 436] for more information (1438, 3/April/2008).
4. **Lotus Notes - Update profile document (new):** The new action can be used to specify the value of a specific field of a database' profile document. See *Script Action: Update profile document* on page 478 for more information (1473, 8/July/2008).
5. **Lotus Notes - Sign/Unsign document (fix):** In previous versions, the action could apparently execute with no error, but the resulting document (adminp request) was not accepted. The error occurred when creating administration requests for Domino version 7 servers. (For version 6.X Domino servers, the problem was not found). The issue has been resolved (1440, 9/April/2008).
6. **Lotus Notes - Update ACL (fix):** The ACE name that is specified as part of the Access Control Entry specification can now hold variables. In previous versions, variables were not replace at run-time by their actual values. (1142, 21/April/2008).
7. **Lotus Notes - UMRA Service fix:** When using an initial project with UMRA Forms projects, the UMRA Service did not properly release the resources used for Lotus Notes databases that were initialized in the initial project. Eventually, this could prevent the UMRA Service and other applications being able to access Lotus Notes databases (1435, 19/March/2008).
8. **Lotus Notes - Create document (enhancement):** The action now exports the document e.g. notes ID of the created document.
9. **Lotus Notes - Register person (advanced) (enhancement):** The action now supports the creation of mail file replicas using property Mail – Mail file replicas (1447, 29/April/2008).
10. **Lotus Notes - Set items (fix):** The order of **text item values** of **text list** items as specified with action **Set items** is now preserved. (1462, 12/June/2008).
11. **Lotus Notes - Example projects (new):** Several UMRA Lotus Notes example projects are added. Almost all of these project show how to setup Lotus Notes administration requests in order to manage Lotus Notes accounts and mailboxes.

**Forms:**

1. **Variable generic table column names (new):** The names of columns in a variable generic table can be specified as variables (%NameColumnA%, %NameColumnB%, etc) (1458, 2/June/2008).
2. **Name of client computer (new):** When a form is submitted by clicking a button, a variable, %UmraClientComputerName%, is generated. The variable holds the name of the client computer and can be used by the UMRA Service. See *Built-in variables* on page 618 for more information (1462, 11/June/2008)

**Database**

1. **Database connection lost when database reset.** With UMRA Forms, an error could occur when databases were reset or restarted. In these situation, the UMRA Service was not able to reconnect to the database unless the UMRA Service was restarted. The problem was caused by an incorrect update of the database connection cache maintained by the UMRA Service (1433).
2. **UMRA Console log with test query (new):** When executing a test query of a table from the UMRA Console application (Setup generic table, Run test, Test...) the query is now written to the UMRA Console log (1458, 2/June/2008).

**Automation**

1. **UMRA Automation 64-bit support (new):** The UMRA Automation software is now available for both 32-bit and 64-bit platforms. As a result, web-pages that are part of IIS web-site can run on 32-bit and 64-bit IIS platforms. See *UMRA COM on 64-bit platforms* on page 73 for more information (1445, 23/April/2008).
2. **Name of automation project (fix):** When creating a new automation project, the new project must now have an unique name. (27/May/2008, 1455)
3. **Automation log files (fix):** The specific automation log file settings, e.g. maximum log files size and maximum number of log files per project are now preserved when the UMRA Service is restarted (1462, 12/June/2008).

**General**

1. **UMRA on Vista and Windows Server 2008 (fix):** All UMRA applications now run on Windows Vista and Windows Server 2008 with no problems (1462, 12/June/2008).
2. **Fix error 20403:** When transferring large amounts of data ( > 1 MB) from either UMRA Forms or the UMRA Console application, an error could occur with error code 20403. This specially happened when variables storing big tables were used. (1433)
3. **Project name variable:** When a project is executed, the name of the project is now stored in a variable. Two variables are used for this purpose: %UmraProjectName% and %UmraProjectNameStack%. Variable %UmraProjectName% contains the name of the (deepest) project that is currently executed. %UmraProjectNameStack% contains the name of all projects:

the deepest child project and all parent projects. See *Built-in variables* on page 618 for more information (1435, 20/March/2008).

4. When a project is executed, the following new UMRA variables are generated:  
%CurrentSystemDate%, %TimeStamp%, %UmraFormSubmitDomain%,  
%UmraFormSubmitUsername%, %UmraPath%, %SystemRoot%. See *Built-in variables* on page 618 for more information (1455, 28/May/2008).
5. **Error importing large amount of projects.** An error could occur when importing large amounts (> 250) of projects in a single operation. The operation has been changed to support more projects (1433).
6. **Distinguished names with slash (/):** The following actions – properties are updated to support distinguished names containing one or more forward slashes: action Set attribute (AD), property Active Directory object LDAP name, action Create object (AD), property LDAP Container, action Set group memberships (AD), property Active Directory name, action Create user (AD), property LDAP Container, action Remove specific group memberships (AD), properties Group name (LDAP) and Account name, action Move cross domain, properties Source object and Target container, action Create contact (AD), property LDAP Container, action Get user table (...), property LDAP path, action Create group (AD), property LDAP Container, action Move – rename (AD), property OU-Container LDAP name, action Get object (AD), property LDAP name (1438, 8/April/2008).
7. **Script error handling (fix):** When the error handling settings of a script action are updated, the project is now marked as dirty (e.g. needs to be saved due to changes in the project) (1458, 2/June/2008).
8. **Log variables - Display of Carriage Return Line Feed:** When a variable value contains carriage-return and/or line-feed characters, the action Log variables will show these characters ([cr],[lf],[crlf]).
9. **Scheduler tab shown (fix):** When an automation project is scheduled to run once, and the run time is passed, the project window now shows the scheduler tab. In previous versions, the scheduler tab was only shown if the project schedule time was in the future (1462, 11/June/2008).
10. **Drop down lists, edit field (fix):** In a number of dialogs and tabbed windows, the the edit field of drop down lists now automatically scrolls in a horizontal direction when text is entered. In previous versions, the length of the entered text was limited (1462, 13/June/2008).

## Version 9.0 Build 1425, February 1, 2008

### New Features

1. **Major new area of functionality:** Support of Powershell. The UMRA software is extended with the Powershell Agent service. The service supports the integration of UMRA and Powershell. New actions that can use any Powershell cmdlets can be added to UMRA in a dynamic manner to extent the functionality of UMRA. For more information, see *Powershell Agent service* on page 2.

2. **Major new area of functionality:** Support of Exchange 2007. Based on the new Powershell Agent service, over 25 new action are added to UMRA to support the management of Exchange 2007 mailboxes and other resources. For more information, see *Exchange 2007* (see "Introduction Exchange 2007" on page 1).

**This new functionality is licensed as a separate module. Existing users should contact Tools4ever for an upgrade of their licence keys when required.**

## Version 8.0 Build 1343, May 4, 2007

### New Features

1. **Major new area of functionality:** Management of IBM's **Lotus Notes** environments. In addition to the existing possibility to manage the Lotus Notes Directory with LDAP, UMRA has been extended to interface with native **Lotus Notes** environments; it can now create, delete and move **Lotus Notes** user's and resources directly, and perform a vast number of other **Lotus Notes** related actions. This functionality is completely integrated with the current functionality of UMRA. For this purpose no less then **25 new UMRA script actions** have been created see *Lotus Notes Actions* on page 392 for an overview.

For a description how to configure umra to start using these actions, see *Configuring the UMRA Console for use with Lotus Notes* on page 37 and *Configuring the UMRA service for use with Lotus Notes* on page 42 .

This new functionality is licensed as a separate module. Existing users should contact Tools4ever for an upgrade of their licence keys when required

2. **New project hierarchy** for server based projects.  
The projects on the server can now be organized in a tree structure. Note that the tree structure is currently only for display purposes, and is visible when the projects listed with the menu option **UMRA Service, Manage server Projects**.
3. **Update the license Mechanism.** Updated the license mechanism to facilitate a new licence that specifically allows the SSRPM (Self Service Reset Password Management) product of Tools4ever to interface with UMRA.
4. **New Action.** A new action *Script Action: Count licensed - domain/OU accounts* on page 371 has been added. Use this action in a script to find out how many user accounts are there in the domain/ou and compare these with the number allowed according to the license. This action allows to find out if a domain is nearing the limit specified in the license. You may want to use this in a scheduled script that sends an email to an UMRA administrator if the license count is almost reached.

## Enhancements

1. **Action enhanced: For Each.** The maximum number of possible variables that can be coupled to the table columns in the called project, has been increased from 20 to 300. (1307-1338)
2. **Action enhanced: If - Then - Else.** The variable argument was only allowed to exist of exactly a single variable. It has been modified so that is is allowed to consist of a text string that contains variables. (1307)
3. **Action enhanced: Move - rename user (AD).** The action has been extended to allow for the move and rename of other objects than users, like for instance **contacts**. Therefore the action has been renamed to **Move - rename (AD)**.(1307)
4. **Action enhanced: Load LDAP Modification Data.** The name of the attribute to modify can now also be specified as a variable, instead of fixed in the action.
5. **Action extended: Manage Table data.** The action is extended with three table data operation functions.
  - Convert multi-text variable to table. Used to convert a variable of the type **text-list** to a table of a single column, and a row for each value in the text-list.
  - Convert multi-value variable to table.
  - Convert table column to multi text variable. Used to convert a specific column of a table to a text-list variable.
6. **Action extended: Fomat variable value.**New formatting functions have been added: Delete: Leading blank characters and Delete:Trailing blank characters
7. **Logging.** Added an option to be able to enable or disable logging to memory for server projects. With the project script tab selected, choose **Actions, Project script properties**, and select the **options** tab to access this functionality. switching it off may significantly increase performance when running projects from the delegation client.(1307)
8. **MySQL datatypes.** For MySql server support is added for the **date** datatype, which contains only the date, not a time. As UMRA itself does only support **date-time**, as a datatype, when reading it will be converted to a date-time value with the time set at 12.00 hr pm.(1307)

## Major Fixes

1. **Generic table.** A memory leak has been fixed which occurred when reading multi-value values in a generic table; for instance when querying the Memberof attribute in an LDAP query. (1303)
2. **Action Create user (AD).** When a user cannot be created due to password complexity requirements, the temporarily created user object is now correctly removed from Active Directory(1307)
3. **Script Action: Move Exchange Mailbox.** A fix has been created for an crash problem introduced by SP2 of Exchange 2003.due to incompatibilities of the updated Exchange software.(1307)
4. **Script Action: Modify exchange mailbox permissions 2000/2003.** A memory leak has been fixed (1307)

5. **Script Action: Move Exchange Mailbox.** Action could cause an exception if the specified user object variable was not valid, for example if it was the result of a failed Get user Action. Now it gives a correct error message and continues with proper error handling as specified.(1311)

#### Minor fixes

1. **Sheduler.** When a project was sheduled for daily execution, the scheduling could not be disabled (it could still be configured as never though). This has been fixed (1309)
2. **Generate generic table-LDAP Query.** The LDAP query filter could not contain variables, they where not resolved at run time. This has been fixed.(1309)
3. **Script Action: Execute script.** The name of the script to execute could not contain variables. now it can. (1311)
4. **Script Action: Update date-time variable.** The action parameters can now contain variables. (1311)
5. **Script Action: Format variable value,** the formating function **Delete, all matching characters, specified as ASCII codes** deleted both upper and lowercase of the specified value when applicable. That is, when specifying 97, in order to delete 'a' accidentally also all occurrences of 'A' where deleted. This has been fixed.
6. **Script Action: copy directory.** When the copy fails because the destination drive is full, UMRA logged a correct error message, but the error handler (the "on error" part of the action) was not invoked. Now the error handler is invoked correctly.(1311)

#### Version 7.6 ,Build 1302, August 23rd, 2006

#### New Features

1. **Project Scheduler.** A major new mechanism to run projects has been added to the UMRA service: the UMRA **Scheduler**. UMRA projects can now be executed automatically at scheduled times by the UMRA service. Open a project and choose Actions-->Scheduler to view and specify scheduling information for the active project. Choose View-->Scheduler to see the scheduling overview of the UMRA Service. Press F1 in those windows for more information.
2. **New action - Join Table Data.** The *Script Action: Join table data* on page 533 has been added.
3. **New action - Manage table data.** The *Script Action: Manage table data* on page 528 is extended with an option to sort a table based on the column name.
4. **Action extended : Generate generic table.** The *Script Action: Generate generic table* on page 527 , is extended with the option to generate a table from a .csv file, and to specify the names of the columns explicitly.
5. **Security improved.** All RPC communication between the UMRA console, UMRA Service,UMRA Com and the UMRA forms client is now encrypted by means of SSL.

6. **Aborting scripts.** A mechanism has been added to see which scripts are currently actively running, and also to make it possible to abort those running scripts if required. choose UMRA Service -> Control Running projects from the menu to go to the overview.
7. **Search and Replace.** A "search and replace" option has been added to search (and or replace) through all configured action property values and specifications in all projects in the active workspace. Useful for instance if you want to rename a variable that is used in a lot of places.

## Enhancements

1. **Service logging.** The service can be configured (choose UMRA Service-->Service properties;Advanced tab) to create user specific cyclic log files rather than general ones. The names of the log files are derived from the name of the account with which the a user connects to the service.
2. **Project logging.** Previously, at the start of each project execution, the values of all variables in the project where logged. This could be a lot of uninteresting data, especially if a project was called many times within a "for each" action. Now this can be switched off in the project options of a specific project (choose Actions-->Project script properties;Options tab)
3. **UMRA Com performance.** UMRA Com is extended with a new interface call (EnableReturnLog) to specify whether or not the log info generated by the service should be returned to the calling Com Object. By default the log information that the called service project generates is returned to the calling Com Object, so that it can be displayed if required. If this option is switched off, no log information is returned to the calling Com object. In both cases the log information is logged in the logfiles on the server. Disabling the return of log information increases the performance.
4. **UMRA command line Application.** The UMRA command line application "UMRACmd.exe", now has an extra command line option (-NoLog), which suppresses log information shown on the command line and increases performance.
5. **Column names in Database Query.** Form tables with database query: When the query is run in the console in the test tab, the column names as implicitly returned by the query are stored , so that they subsequently can be used with the specification of the columns to show.
6. **Action enhanced: Manage table data.** The *Script Action: Manage table data* on page 528: Within the sub-option "Export to .csv", it is now optionally possible to write a header with the column names to the file.
7. **Extra variables in automation projects.** The %NowDay%, %NowYear% default variables are now also available in automation projects.
8. **Console GUI:** When creating a new project, the project name is now first proposed, before being used.
9. **Extra output variables in script actions.** The actions *Script Action: Create User (AD)* on page 3, *Script Action: Create group (AD)* on page 138 and *Script Action: Create object (AD)* on page 117 have been extended, so they will now optionally output the Object Distinguished Name of the object they have just created, to a variable. In larger scripts, this name was often needed further on in the



script. Previously it had to be explicitly retrieved by using for example the "Get attribute" script action. and that is now no longer required.

10. **Action enhanced: For-Each.** The *Script Action: For-Each* on page 572 is extended with the possibility to specify which variables should be available to use for the called project. Making unavailable variables that are not used in the called project can drastically (> 10 times faster) improve performance.

### Major Fixes

1. **Table in forms Client.** When a selection was made in a table, clicking on a column header to resort the table would change which items were selected. This has been fixed.
2. **Logging mechanism.** A potential critical error in the internal working of the logging mechanism on the service has been fixed. In some circumstances, when the execution of form script did not finish within the timeout period, and several other conditions were met, an error in the cyclic log mechanism could cause a critical failure in the UMRA service. This has been fixed.
3. **Script action Update date-time variable.** The *Script Action: Update date-time variable* on page 552 did not work if the option "convert to 100ns interval since 1-1-1601" was chosen. This has been fixed.

### Minor Fixes

1. Service logging - Several log files use a method of cyclic logging; that is, when a file *log1* has reached a specified limit, logging continues in file *log2*, etc. In some circumstances a few log messages that were generated at the time of file change, were not correctly written to file. This has been fixed.
2. LDAP name syntax. If the name-part of an object contains a forward slash "/", the format that LDAP uses to specify the name of the object was not handled correctly by UMRA. This has been fixed.
3. Script action "Manage table data", option "export to .csv file". The output file contained incorrect "new line" like characters. When read in certain applications (MS Access for example), this would result in the display of additional empty lines. This has been fixed.
4. The Action "Set variable" did not write a message to the log when it was executed. Now it does.
5. When creating a completely new name generation algorithm, it was not created with the specified name, but with a number. This has been fixed.
6. If a workspace contains more than 1 project, and has been modified, the user will be prompted to save the workspace when exiting the console.
7. A project is now also marked dirty when a script action is removed from the script. So the user is correctly prompted to save when exiting.
8. The next issue only occurred when using a forms project containing an initial project with a "Generate generic table" script action. The contents of this table can be stored in a variable and

shown afterwards in a Form Table. The order of the entries as shown in the table was not the same as the order of the entries in the original file. This has been changed, so that, if no sorting method is selected, the original order of entries read from the file is maintained.

9. When using the *Script Action: Generate generic table* on page 527 to perform an LDAP search query, the specified time-out values were not correctly used. For those queries which take a very long time to return the result, this could cause the search to fail. This has been fixed.
10. The Ctrl-C and Ctrl-V copy and paste actions now also work for the static text and input text fields in the forms client
11. The Set variable action has been modified. If the current value of a variable contains a reference to itself, it is no longer resolved even when the option "resolve now" is on.
12. When a project is opened, and closed immediately thereafter, it is now added to the list of recent projects. Previously, it was not added.
13. In *Script Action: Create Exchange Mailbox (2003/2000)* on page 156. The name of the variable used for the User Object can now be specified. Previously this was always fixed to %UserObject%. and thus the action could not be used if the user object was stored in a different variable.

#### Cosmetic Fixes

1. The icons in the main button bar corresponding to the project tabs (script, form ,preview, network data etc), are now checked/unchecked depending on the existence of the associated project tabs in the active project.
2. Action format variable,sub-action Replace substring with ASCII codes. The description field has been enlarged.
3. Some irregularities in the display after vertically scrolling a form in the Forms Client have been fixed.
4. The UMRA log file UMRASvcFormLog.txt contained superfluous extra new line characters between log messages. This has been fixed.

#### Build 1263, May 12th, 2006

#### New features

1. **Workspace** - A new feature is the introduction of workspaces. This allows the user to structure complex projects more efficiently. For more information, see *UMRA workspace* on page 770. (1263)
2. **GUI** - The main interface has been drastically improved. The key project concept of UMRA has always been the creation of an UMRA project script using input data from various sources ((CSV) files, databases, other applications). The main interface has been reworked to support this concept more strongly. See also *UMRA Basics* on page 3 for more information on UMRA's basic concepts. Depending on the kind of solution you want to create (MASS, Forms, Automation),

UMRA will include the relevant project components for the selected project. Also, as part of the overall interface makeover, the Personal Assistant, What's this help and Project Wizards have been removed. (1263)

3. **IMPORTANT - All MASS projects created in previous versions of UMRA (.UPJ files) have to be upgraded.** Please follow the procedure as described in *Upgrading MASS projects from previous versions* on page 50 to upgrade these files. (1263)
4. **Copy directory script action** - when using this script action, the date, time and attribute stamps are now copied as well. (1263)
5. NETBIOS name of the computer accessing the **UMRA Service** is now stored in the UMRA variable %UmraClientComputer%. (1263)
6. **Remove SID History script action** - A script action has been added to remove the SID history of a (removed) user account. See *Script Action: Remove SID history* on page 130 for more information. (1263)
7. **Rename directory service object (LDAP) script action** - UMRA contains several script actions to manage LDAP directory services. A script action has now been added to change the distinguished name of an entry in the directory service (only available for LDAP version 3). See *Script Action: Rename directory service object (LDAP)* on page 390 for more information. (1263)

#### Major fixes

1. **Setup LDAP session** - in the previous UMRA version, LDAP version 3 was not correctly initialized. LDAP sessions exist in version 2 and 3. Version 3 offers more functionality, which is why UMRA initializes the session based on version 3. This was incorrectly implemented, resulting in an error on Windows 2003. This has been fixed. (1263)

#### Minor fixes

1. **Command line tools:** In previous versions, the port option of the command line did not work. This has been fixed. (1263)
2. **Modify exchange mailbox permissions (2000/2003):** In previous versions, an error in a scenario where you have users A, B and C. User A has a mailbox and user B gets permissions for this mailbox. If user B was removed from Active Directory and a third user granted permissions to the mailbox of user A, an error would occur. This has been fixed. (1263)
3. **Set user group memberships (AD)** - In some rare situations, the group selections were unexpectedly cleared. This has been fixed. (1263)
4. **UMRA service** - In previous versions, if you had changed the data cache retry value, this value would be reset to the default value if the UMRA service was restarted. This has been fixed. (1263)

### Cosmetic fixes

1. **Execute script script action:** The combo box for the **Project** property has been enlarged to display long path names. (1263)

### Build 1227, January 6th, 2006

#### Major fixes

1. **Setup Security** - All directories and files for which security was set (using either propagation mode or not), were made read-only. This has been fixed. (1227)

### Build 1225, December 23rd, 2005

#### New features

1. **Script actions to manage LDAP directory services:** *Script Action: Setup LDAP session* on page 383, *Script Action: Load LDAP modification data* on page 386, *Script Action: Delete directory service object (LDAP)* on page 389, *Script Action: Add directory service object (LDAP)* on page 387, *Script Action: Modify directory service object (LDAP)* on page 388, *Script Action: Search LDAP* on page 391. Examples of such directory services are Novell eDirectory, Linux OpenLDAP and Microsoft's Active Directory. All these directory services can now be managed using UMRA. (1225)  
  
For general information on using UMRA for managing LDAP directory services using UMRA, see *Managing LDAP directory services using UMRA* on page 25.
2. **New action - Delete attribute value (AD):** Script action has been added to delete a specific attribute value. For more information, see *Script Action: Delete attribute value (AD)* on page 129. (1225)
3. **New action - Execute script:** Script action has been added to execute the script of another project and merge the updated variables into the current project. For more information, see *Script Action: Execute script* on page 571. (1225)
4. **New action - Generate password:** Script action has been added to generate a password. For more information, see *Script Action: Generate password* on page 565. (1225)
5. **New action - Get user table:** Script action has been added to generate a table containing information regarding locked-out users, disabled users and other user account control states. For more information, see *Script Action: Get user table (AD)* on page 51 (1225).
6. **New action - List files and/or directories:** Script action has been added to obtain a list of files and directories in a variable. This variable can in turn be displayed in a form table. For more information, see *Script Action: List files and/or directories* on page 367. (1225)

7. **New action - Set encrypted variable:** Script action has been added to set the value of the specified variable to the encrypted value of the entered text. For more information, see *Script Action: Set encrypted variable* on page 546. (1225)
8. **Enhanced - Check form input:** An option has been added to check the string length of text which has been entered by the user in an input box. The variable contents check specification has been restyled. For more information, see *Form action - Check form input* on page 634. (1225)
9. **Enhanced - Form property Initial Project:** The Initial Project tab has been renamed to Execution control and now includes an option to set a time-out for script execution. (1225)
10. **Enhanced - Manage table data script action:** Option has been added to remove rows and columns of a table. (1225)
11. **Enhanced - Return project form:** Option has been added to specify a variable for the returned project form. (1225)
12. **Enhanced - Table form field options:** An option has been added which can be activated to execute a default button when the user doubleclicks a table entry. For more information, see *Table form field - Options* on page 659. (1225)

#### Major fixes

1. **Manage exchange recipient mail addresses (2000/2003) script action:** In previous versions, UMRA would crash if a non existing AD had been specified. This has been fixed. (1225)

#### Minor fixes

1. **Edit User (AD) script action:** In previous versions, the on error actions did not work for this script action. This has been fixed. (1225)
2. **Execute service command script action:** The text in property **Wait for status completion** has been changed from "If set, the action will not complete until the services reports the requested state or the time-out period is expired." to "If set, the action will not complete until the service has the requested state or if the time-out period has expired." (1225)
3. **For-Each script action:** In previous versions, when a network location was specified in the **Script project** property, the path of the standard project location was included, which resulted in an execution error. This has been fixed (1225).
4. **Form elements - Display tab:** In previous versions, the option "Vertical pixel offset" would only work for text fields. It now also works for tables, checkboxes, radio buttons and standard buttons. (1225)
5. **Generate name(s) script action:** In the previous versions, the text under the **Iteration** property read "can only be used for mass projects". This is incorrect. It can also be used for Forms projects in a loop. This text has been removed. (1225)

6. **Generic table type Variable:** In previous versions, when you specified the table height in the **Options** tab, this specification would not be used if the variable did not contain any items. This has been fixed. (1225)
7. **LDAP attribute variables:** In previous versions, when the LDAP query of an existing LDAP Search was changed, any variables assigned to these attributes would disappear. In the new build, the variable specification remains intact when you add attributes to your LDAP search. When you change the order of appearance however, the mapping will no longer be correct. This is by design. (1225)
8. **Licensing:** In previous versions, when a DNS domain name had been licensed, UMRA would not accept a DNS name for the domain. This has been fixed. (1225)
9. **User name generation algorithm:** In the previous version, if you specified an "Arbitrary sequence" (e.g. "01,02,03") and clicked OK and then opened the iterator again, the iteration object had changed to "01, 02, 03" (spaces added). These spaces would also end up in the user account name. Each subsequent cycle would add another space. This has been fixed. (1225)

#### Cosmetic fixes

1. **Table columns:** Table columns are now always dynamically resized when the table dimensions are adjusted. (1225)
2. **Project forms display name:** Project forms now also have a display name. By default, the display name is identical to the project name and is also shown in the Forms project client tree. (1225)
3. **Project execution status info:** When a form project is processed by the UMRA Service which takes a long time to complete, the message "Please be patient while your request is being processed" is displayed. (1225)

#### Build 1201, September 30th, 2005

#### New features

1. **New action - Configure service:** Script action has been added to configure Windows computer services. To be used in combination with the new script action *List services status* on page 373. For more information, see *Script Action: Configure service* on page 379. For an example project on how to use these script actions, see *Managing Windows computer services* on page 21. (1201)
2. **New action - Convert value of variable:** Script action added to convert the value of a variable (logical AND, large integer to date-time or large integer to specified text). For more information, see *Script Action: Convert value of variable* on page 554. (1201)
3. **New action - Delete object (AD):** Script action has been added to delete an existing Active Directory object. For more information, see *Script Action: Delete Object (AD)* on page 119. (1201)
4. **New action - Edit share:** Script action has been added to edit an existing share. For more information, see *Script Action: Edit share* on page 364. (1201)

5. **New action - Execute print job command:** Script action has been added to start, resume, pause or delete a print job. To be used in combination with the new script action *List printer documents* on page 380. For more information, see *Script Action: Execute print job command* on page 382. For an example project on how to use these script actions, see *Managing printer queues* on page 77.
6. **New action - Execute service command:** Script action has been added to manage services, to be used in combination with the new script action *List services status* on page 373. For more information, see *Script Action: Execute service command* on page 377 and *Script action: List services status* on page 373. For an example project on how to use these script actions, see *Managing Windows computer services* on page 21. (1201)
7. **New action - Get terminal services user settings:** For more information, see *Script action: Get terminal services user settings* on page 111. (1201)
8. **New action - Get user info:** Script action has been added to retrieve specific flags of the userAccountControl bitmask attribute (e.g. Account disabled" and "User must change password at next logon" ). For more information, see *Script Action: Get user info* on page 101. (1201)
9. **New action - List printer documents:** Script action has been added to collect the printer documents for a specific printer. To be used in combination with the new script action *Execute print job command* on page 382 to manage printer queues. For more information, see *Script Action: List printer documents* on page 380. For an example project on how to use these script actions, see *Managing printer queues* on page 77. (1201)
10. **New action - List services status:** See item 6. (1201)
11. **New action - Move cross domain:** Script action has been added to move a user account to another domain. For more detailed information, see *Script Action: Move cross-domain* on page 67. For an example project on how to use this script action, see \$\$\$\$. (1201)
12. **New action - Move Exchange mailbox:** Script action added to move an Exchange mailbox. For more information, see *Script Action: Move Exchange mailbox* on page 167. (1201)
13. **New action - Update database:** Script action added to update the record set of an existing database. For more information, see *Script Action: Update database - introduction* on page 539. (1201)
14. **New action - Update date-time variable:** Script action has been added to performs basic numeric operations on date-time variables (e.g. to return the current date). For more information, see *Script Action: Update date-time variable* on page 552. (1201)
15. **Enhanced - Checkbox form element:** Added the option to set the initial state of a checkbox. For more information, see *Checkbox form field* on page 640. (1201)
16. **Enhanced - database support for all database types:** For more information, see *Database setup - Other databases* on page 628. (1201)
17. **Enhanced - Default service log file:** Increased from 1 MB to 20 MB. (1201)
18. **Enhanced - Error messages:** Error messages for database actions have been enhanced to get better insight into errors returned for database queries (update and select queries). (1201)

19. **Enhanced - For-Each:** Added a break condition for a For-Each loop. (1201)
20. **Enhanced - Format variable value:** Four formatting options added to search and replace for a string ending or starting with a specific substring. (1201)
21. **Enhanced - Form export:** Illegal file name characters are now replaced by underscores. In previous version, exporting forms did not work if a title included a colon (":"). (1201)
22. **Enhanced - Form projects:** Before a form is shown, the script of another form project can be executed by specifying an initial project. See also *UMRA Project Properties - Form Options* on page 767. (1201)
23. **Enhanced - General:** Shortcut keys have been added for Open (CTRL + O), Save as (CTRL+shift+S) and Close CTRL+W (all related to projects). (1201)
24. **Enhanced - Generic table:** Added the possibility to use temporary variables for testing purposes. Variables can be specified which will be filled during run-time during the test. (1201)
25. **Enhanced - Generic table - Table type "Variable" added:** Using this table type, the content of a variable can be used as a generic table. For more detailed information, see *Viewing data from Active Directory, LDAP, databases* on page 9. (1201)
26. **Enhanced - Get Object (AD):** class name, path, parent path, schema and guid properties can now be saved as a variable. (1201)
27. **Enhanced - Generic table - Database query:** The data type Memo in MS Access is now supported as well. (1201)
28. **Enhanced - Generic table - Exclusions:** In the previous version a message was displayed in the delegation client and the UMRA service log file if the service or console application could not manage to determine the members of an excluded global group. This is now also displayed in the console application if the preview window is constantly updated (see # 30 as well). (1201)
29. **Enhanced - Generic table - Exclusion tab:** Doubleclicking a global group now starts the edit function. (1201)
30. **Enhanced - Generic table - error messages:** If a database query returns a data type which is not supported by UMRA, the error message is only displayed for the first row. Note that this also applies to a situation where a table contains one row with multiple unsupported data type fields. (1201)
31. **Enhanced - If-Then-Else script action:** now includes the option to check if a text-list type variable contains a specific text. (1201)
32. **Enhanced - Log window:** Log Window now supports scrolling for longer messages.
33. **Enhanced - Manage table data:** Option has been added to add tables (assuming the number of columns in both tables is identical). (1201)
34. **Enhanced - Manage table data:** Option added to export table to a CSV file. (1201)
35. **Enhanced - Radio buttons:** Initial state of a radio button can now be set according to a variable. (1201)



36. **Enhanced - standard name generation algorithms:** these can now be pushed to any UMRA project. (1201)
37. **Enhanced - UMRA forms:** option added to specify if the form content, as shown in the preview, should be constantly updated during form design. See also *UMRA Project Properties - Form options* on page 767. (1201)
38. **Enhanced - Update numeric variable:** Several options added to perform basic calculations on numeric variables. For more information, see *Script Action: Update numeric variable* on page 550. (1201)
39. **Change - Form projects - Preview window:** The preview window in the console is now generated by the UMRA service. In the previous versions, the preview window was generated by the console (only the script was executed by the service). (1201)
40. **Change - General - Project timeout:** In the previous version, if a form was submitted in the client (in preview) and the cache was still empty, a timeout was given when the project could not be executed within 60 seconds. This timeout has been increased to 300 seconds. (1201)
41. **Change - Method of counting the number of user accounts:** this has changed for dns/ou license keys. Accounts ending on "\$" (computer accounts) are no longer included in the user count. For netbios keys nothing has changed. See also *Network bar - Count users* on page 728. (1201)
42. **Change - Tree structure for script actions:** Variable actions have now been placed in Variable actions folder with subfolders for Variable operations, Table, Database, Name generation, Programming and Mail. (1201)

### Major fixes

1. **Create user script action:** In previous versions, if the password of a newly created user did not meet the complexity requirement, an iteration occurred. As a result, the user was created 100 times. This has been fixed. (1201)
2. **Exporting Form projects:** In previous versions, file was exported even when the "Cancel" button had been clicked. This has been fixed. (1201)
3. **For-Each script action:** In the previous version, the label assigned to the For-each script action was no longer present if the project was saved, closed and opened again. Jumping to the label was then no longer possible. This has been resolved. (1201)
4. **Fomat variable value script action:** In the previous version, if a comma was replaced with a string which also included a comma, this comma would in turn also be replaced, resulting in a loop. This has been fixed. (1201)
5. **Generic table - Database queries:** In the previous version, an error message was displayed when a query retrieved data of an unsupported data type and the remaining fields would be placed in the wrong columns. This has been fixed. (1201)
6. **Generic table - Database queries:** In the previous version, no values were returned if the query included two fields containing unsupported data types. (1201)

7. **Generic table - Database queries:** In the previous version, numeric and autonumber data fields were interpreted as text, leading to an incorrect sort order in UMRA forms. This has been fixed. (1201)
8. **Generic table - Database queries:** In the previous version, Date-time fields were not retrieved correctly. This has been fixed. (1201)
9. **Script action Generate generic table:** binding to a variable. In the previous version, if the script action was executed several times, the binding variable would be overwritten by the actual value. (1201)

#### Minor fixes

1. **Create User script action:** Logon hours property. If the current settings were edited and cancelled, the changed logon hours would appear if you clicked the Edit button once more. This has been fixed. (1201)
2. **Form project - Minimize window:** In the previous version, if the window had been minimized, this setting would not be undone after restarting the console. This has been fixed. (1201)
3. **General:** If a script action was deleted in the previous version, the window focus was not automatically set to the next script action (e.g. to perform another delete operation). This has been fixed. (1201)
4. **Generate generic table script action:** In the previous version, the name of the output variable for the generic table did not appear in the combo box for other script actions. This has been fixed. (1201)
5. **Generate random number:** In previous versions, if the Generate random number action appears twice in a script, with an identical range, the same value was returned when the script was executed (in other words, the number was not entirely random). This has been fixed. (1201)
6. **Generic table - lastLogon attribute:** In the previous version, if the last logon data for a user account were retrieved through a generic table, the data were sorted on time instead of date. This has been fixed. (1201)
7. **Generic table--->LDAP Query--->Attributes Tab-->"Delete ALL" button:** In the previous version, this button only worked if entries in the attributes list had been selected. It now also works if this is not the case. (1201)
8. **Get attribute script action:** In the previous version, the list of values in the LDAP attribute display name property did not include "memberOf". This has been fixed. (1201)
9. **Get object script action:** In the previous version, no error message was displayed when a variable name was entered which was not specified between percentage signs (%<NameVariable>%.). This has been fixed. (1201)
10. **If-Then-Else script action:** In the previous version, if a label was specified for the Else condition and then disabled by deselecting the Else checkbox, the Else condition would still be executed. This has been fixed. (1201)

11. **Log Window:** the log window now supports scrolling for longer messages. (1201)
12. **Logging variables:** When logging variables containing large text arrays (e.g. the multi text result of an LDAP search action), the log now displays the number of text elements. In previous versions, this action resulted in an "Invalid error string" message in the log file. (1201)
13. **Mass projects, Generate Generic Table, test page:** In the previous version, the option "Run test on UMRA service" was available even though Mass does not use the UMRA service. This has been fixed. **Send mail message action** - In the previous version, "%Username%" was shown in the log file instead of the resolved value. This has been fixed. (1201)
14. **Manage table data script action:** In the previous version, the field Cell data value for the option Set the data for the specified row and column of the table only accepted a string of a limited pixel length. This has been fixed. (1201)
15. **Search Object script action:** In the previous version, the property values for this script action were not shown correctly in the log file (the action itself was executed correctly). This has been fixed. (1201)
16. **Send mail message script action:** In the previous versions, the "X-Sender:" field could not be edited. This has been fixed. (1201)

#### Cosmetic fixes

1. **Log variables script action:** In the previous version, the name of this script action changed to ""Write the current script variables in the log" when it was dragged to the script action window. This has been fixed. (1201)
2. **Generic table - Query tab:** In the previous version, the vertical scrollbar was not shown. This has been fixed. (1201)

#### Build 1164, July 1st, 2005

#### New features

1. **Form project - Generic table:** A new form field has been introduced. This form field allows you to display data from Active Directory and user information stored in other information systems and to use these data as input for an UMRA project script. For more detailed information, see *Viewing data from Active Directory, LDAP, databases* on page 9. (1164)
2. **New action - For-Each function:** The For-Each function evaluates the rows of a table and executes a script for each row which is defined in another project form. This script action is created in <Project form1> whereas the action which needs to be executed as a result of the For-Each action is created in <Project form2>. This way, you can reuse complex For-Each constructions in other projects. For more information, see *Script Action: For-Each* on page 572. (1164)

3. **New action - Generate generic table:** The generic table has also been made available as a script action. Unlike the generic table which is used in a form, the data are not shown in a form table, but used directly as input for an UMRA project script. See also *Viewing data from Active Directory, LDAP, databases* on page 9. (1164)
4. **New action - Get primary group:** This script action retrieves the primary group of the user. For more information, see *Script action: Get primary group* on page 154. (1164)
5. **New action - If-Then-Else function:** This script action has been introduced to be used in a project script to evaluate a condition. This new script action makes it possible for instance, to verify the last logon time of a user and to execute a certain action if the last logon time was more than an X number of months ago. For more information, see *Script Action: If-Then-Else* on page 570. (1164)
6. **New action - Manage table data:** Allows you to manipulate data in an existing table or to create a new table. For more details, see *Script Action: Manage table data* on page 528. (1164)
7. **New action - Merge multi-text variable values:** Merges Variable1 and Variable2 . For more details, see *Script Action: Merge multi-text variable values* on page 559. (1164)
8. **New action - Rename file or directory:** With this action you can rename files and directories and move files to another volume. In previous versions this was only feasible by using the Copy directory, Delete directory and Execute command line script actions. For more information see *Script Action: Rename file or directory* on page 352. (1164)
9. **New action - Remove group member:** This action has been added to remove the group member from a specific group. For more details, see *Script Action: Remove group member* on page 94. (1164)
10. **New action - Remove specific group memberships (AD):** This script action allows you to remove specific group memberships. So far it was only possible to remove all group memberships using the action Remove user group memberships (AD). For more information, see *Script Action: Remove specific group memberships (AD)* on page 137. (1164)
11. **New action - Send mail message:** This script action allows you to send an e-mail message as a result of a previous script action. For more information, see *Script Action: Send mail message* on page 575. (1164)
12. **New action - Set primary group (AD):** This script action is only of interest for those customers who need to change the primary group. This is the case when there are any users who log on to the network from a Macintosh client or who run POSIX-compliant applications. For more details, see *Script Action: Set primary group (AD)* on page 155. (1164)
13. **New action - Update numeric variable:** With this new function you can increment the value of a variable. For more details, see *Script Action: Update numeric variable* on page 550. (1164)
14. **Action - Delete directory:** A property has been added to ignore errors which are generated when the action is executed. (1164)

**Critical fixes**

1. **Name generation algorithm:** When a name generation algorithm contains an endless loop or when no unique names are generated by the algorithm, the UMRA software will now end the execution of the algorithm after a number of iterations. (1164)

**Major fixes**

1. **Create directory script action:** When the directory name is changed to make it unique, the new unique name is now exported. In previous versions, the incorrect already existing directory name was exported. (1164)
2. **UMRA Forms:** When the Control key is pressed while working with a table, the vertical scroll position of the table is no longer changed. In previous versions, the table was scrolled to the first entry of the list. (1164)
3. **Get attribute (AD) script action:** If the property is specified to get a multi-value attribute, the output Attribute value will always contain a multi text list, even if there are no values or just a single attribute value found. (1164)

**Minor fixes**

1. **Search object script action:** In the properties pane of a project window the properties values shown for properties Error if nothing found and Error if multiple found are now correct. (1164)
2. **UMRA Service:** When the UMRA Console or UMRA Automation software build number do not correspond with the build number of the UMRA Service, you are now forced to up- or downgrade the UMRA Service. In previous versions, you could continue to open a project. (1164)

**Cosmetic fixes**

1. **Manage Exchange recipient mail addresses (2000/2003) script action:** The default variable input name is changed from %AdObject% to %ActiveDirectoryObject%. (1164)

**Build 1141, April 29, 2005****New features**

1. **UMRA Automation:** A new module is introduced. The module supports the integration of the functions of User Management Resource Administrator with other products that are used to manage employee and user accounts. See \$\$\$ for more information. (1141)
2. **Delegation:** With UMRA, you can delegate control to helpdesk employees. See \$\$\$ for more information. (1117)
3. **Command line startup:** Run a project automatically with UMRA console when the application is started. See *UMRA Console - Command Line Options* on page 763 for more information. (1141)

4. **Network tree:** For all Active Directory objects, all properties can be shown and managed from the UMRA console application. (1117)
5. **Action - Set attribute (AD):** A property is added to prevent the action from updating the user attribute if the new attribute value is empty. See *Script Action: Set attribute (AD)* on page 124 for more information. (1117)
6. **Action - Get attribute (AD):** The action now supports multi-values and all Active Directory objects, not only users. See *Script Action: Get attribute (AD)* for more information. (1141)
7. **Action - Modify Exchange mailbox permissions (2000/2003):** With this action you can add and remove permission for Exchange mailboxes. See *Script Action: Modify Exchange mailbox permissions (2000/2003)* on page 162 for more information. (1117)
8. **Action - Set Variable:** An option is added to specify when (other) variable names specified as part of the variable value must be resolved. See *Script Action: Set Variable* on page 544 for more information. (1141)
9. **New action - Create contact (AD):** Create contact accounts in Active Directory. See *Script Action: Create contact (AD)* on page 21 for more information. (1117)
10. **New action - Edit user logon:** Reset passwords and manage logon properties of user accounts. See *Script Action: Edit user logon (AD)* on page 47 for more information. (1117)
11. **New action - Modify Exchange mailbox permissions:** Setup the permissions for new or existing Exchange 2003/2000 mailboxes. See *Script Action: Modify Exchange mailbox permissions (2000/2003)* on page 162 for more information. (1117)
12. **New action - Manage Exchange recipient mail addresses:** Setup mail addresses for Exchange 2003/2000 mail recipients. See *Script Action: Manage Exchange recipient mail addresses (2003/2000)* on page 169 for more information. (1117)
13. **New action - Dial-in user settings:** Specify dial-in and VPN settings for user accounts. See *Script Action: Dial-in user settings* on page 114 for more information. (1117)
14. **New action - Set group membership (AD):** Set the Active Directory group memberships for user accounts and other Active Directory objects. See *Script Action: Set group membership (AD)* on page 135 for more information. (1117)
15. **New action - Create group (AD):** Create a group in Active Directory. See *Script Action: Create group (AD)* on page 138 for more information. (1117)
16. **New action - Get object (AD):** Access any Active Directory object to set and read properties. See *Script Action: Get object (AD)* on page 145 for more information. (1117)
17. **New action - Create share:** Create a share on a directory and setup the share properties including security settings. See *Script Action: Create share* on page 361 for more information. (1117)
18. **New action - Delete share:** Deletes a share from a directory. See *Script Action: Delete share* on page 366 for more information. (1141)

19. **New action - Convert to multi-value variable:** Manage values of variables to be converted to multi-value values. See *Script Action: Convert to multi-value variable* on page 556 for more information. (1117)
20. **New action - Manage multi-text value variable:** Manage values of multi-value variables. See *Script Action: Manage multi-text value variable* on page 558 for more information. (1117)
21. **New action - Generate random number:** Generate a random number and assign the value to a variable. See *Script Action: Generate random number* on page 564 for more information. (1141)
22. **New action - Generate name(s):** The name generation algorithms can now be used as a separate action. See *Script Action: Generate name(s)* on page 542 for more information. (1141)
23. **New action - Convert text to date/time:** Convert a text value to a date/time value. Both values are stored in a variable. The method used to convert the text to a date/time value can be specified. See *Script Action: Convert text to date/time* on page 555 for more information. (1141)
24. **New form action - Return other form:** When a submit button is pressed in a form, another form can be returned. See *Form action - Return other form* on page 638 for more information. (1141)
25. **New form action - Iteratively execute project script:** Execute the project script for each item selected in a table. See *Form action - Iteratively execute project script* on page 636 for more information. (1141)
26. **Name generation algorithm:** The configuration of the name generation algorithm is now always stored in the action that uses the algorithm, e.g. the actions *Script Action: Create User (AD)* and *Script Action: Create User (no AD)*. In previous versions, the algorithm could be reloaded from configuration files each time a project was executed. See *Name Generation: Embedded algorithms* for more information. (1117)
27. **Export Variables script action:** The name of the export file can contains date related variables. See *Script Action: Export Variables* on page 559 for more information. (1117)
28. **Export Variables script action:** For multi-value variables, a value separator character can be specified. See *Script Action: Export Variables* on page 559 for more information. (1141)
29. **Create directory script action:** When creating a share for the new directory, the maximum number of connections for the share can now be specified. (1117)
30. **Name generation algorithm:** Methods can now be copied to make it more easy to create similar name generation methods. (1141)
31. **Form project:** The type of popup messages that must be shown when a form is submitted can now be configured. See *UMRA Project Properties - Form options* on page 767 for more information. (1141)
32. **Logon hours:** The action to create and edit user accounts, both in Active Directory and NT, now supports user account logon hours. See *Script Action: Create User (AD)* on page 3, *Script Action: Edit user (AD)* on page 37, *Script Action: Create User (no AD)* on page 68 and *Script Action: Edit user (no AD)* on page 79 for more information. (1141)

33. **Formatting functions:** A function has been added to replace a substring in a text string with ASCII codes. With this function, a user defined control sequence (for example \n) can be converted to a carriage return - line feed sequence (13,10). (1141)
34. **UMRA Console:** The drag- and drop and cut-copy-paste functions have been extended. (1141)
35. **UMRA Service:** A new variable is automatically generated and updated when a form is submitted: %UmraFormSubmitAccount%. The variable contains the name of the user account that submitted a form. See *Built-in variables* on page 618 for more information. (1141)

#### Critical fixes

1. **Demo version:** The demo version now supports all script actions. (1117)

#### Major fixes

1. **Edit user (AD) script action:** The attribute of a user account can now be cleared. In previous versions, an error occurred when the attribute was set to an empty value. (1117)
2. **Move - rename user (AD) script action:** When renaming a user account, the new name can contain comma's (,). (1117)
3. **Create directory - Copy directory script actions:** When setting the permissions of the target directories and files, in previous the versions, the specification of the Read permission incorrectly granted the Delete access right as well. The problem is fixed. (1117)
4. **Create directory script action:** When specifying the permissions for a share, an account can now be removed for the share from the list with permissions. (1141)
5. **Delete directory script action:** The script action now also delete files and directory that start with a dot (.). (1117)
6. **Format variable value script action:** A specification problem regarding the format functions and format function arguments has been resolved. (1141)
7. **Export Variables script action:** The variables can now be exported in UNICODE format. See *Script Action: Export Variables* on page 559 for more information. (1141)
8. **Variables:** The special variables %NowMonth%, %NowDay% and %NowYear% can now be used in all modules (not only in mass projects). See *Built-in variables* on page 618 for more information. (1141)
9. **Mass projects - input data:** The maximum number of columns read from a CSV file has been increased from 26 to 75 columns. (1117)

#### Minor fixes

1. **Edit user (AD) script action:** When a property is specified is an text with no length, the property value is shown as <empty text>. In previous versions, nothing was shown for the empty value. (1117)



2. **Menu option - Add action to script:** The menu option can be used for mass and form projects. (1141)
3. **Menu option - File, Save:** The shortcut key combination Ctrl+S can be used to save projects. (1141)
4. **Get User (AD) script action:** When a user account is specified using a domain name, OU-name and common name (full name), a warning is now displayed if the user account cannot be found and no (empty) OU-name is specified. (1141)
5. **Format variable value script action:** The names of the formatting functions are updated. (1141)
6. **Mass projects - column variable:** When a variable is associated with a column, the name of the variable can now be selected from the list with variable names when specifying script action properties. In previous versions, these variables could be specified but were not shown on the list with variable names. (1141)
7. **Form projects - script message:** When submitting a local form from the UMRA Console application, the script message was shown twice. The problem has been resolved. (1141)
8. **Form projects:** When the form project properties (format, fonts, options and security) are updated, the project is now marked as changed. (1141)
9. **Form project:** The calculation of the length of a form is now more accurate. This results in better vertical scroll bar settings in a form. (1141)
10. **Form project:** The mouse scroll-wheel is now supported in a form. (1141)
11. **Form project:** When changing the table type of a form network table, the columns are now updated. (1141)
12. **UMRA Console:** When the application is closed and a project has not been saved, you can now cancel the application close operation. (1141)
13. **UMRA Console:** When the error settings or label of a script action is specified, the project is now marked as changed. (1141)
14. **UMRA Console:** When dragging and dropping a script action on the same position now nothing happens (as expected). In previous versions, the actions was incorrectly moved to the last position in the script. (1141)

#### Cosmetic fixes

1. **Form projects:** When form projects are opened to be designed, the column width is automatically updated. (1141)
2. **Form projects:** When editing form fields and the Cancel button is pressed, the window no longer indicates that something is changed. (1141)
3. **Tooltips:** The What's This tooltips for form projects are now correct. (1141)
4. **Tooltips:** The tooltips shown in various tree windows are hidden when the mouse is moved in the tooltip area. (1141)

5. **Create directory script action:** The action now logs test only when the script action is executed in test mode. In previous versions, the test only phrase was not shown for this action. (1141)
6. **Create user (AD) script action:** The action now logs test only more explicitly when the script action is executed in test mode. (1141)
7. **All actions - Properties:** When selecting a variable as the new value for a property, the name of the variable is now immediately inserted as the property value if the property value is empty. In previous versions, you always needed to press the Insert button. (1141)
8. **Menu:** In rare occasions, the menu text shown could be updated incorrectly in previous versions. The problem has been fixed. (1141)
9. **Icons:** The icons for the script actions to delete a share and execute a command line are updated. (1141)

## Build 1065, September 17, 2004

### New features

1. **Terminal Services Support:** The new version supports the configuration and specification of Terminal Services settings for new and existing user accounts. (1033)
2. **Name generation:** A new function has been added to add characters at the end of a name to lengthen the name. (1065)
3. **Add account to local group script action:** A new action has been added to add user and global group account to local groups of domains, member servers and workstations. See *Script Action: Add account to local group* on page 90 for more information. (1065)
4. **Create user (AD) script action:** The property **Computer account** has been added to allow the creation of workstation - computer accounts. See *Script Action: Create User (AD)* on page 3 and *Script Action: Create User (no AD)* on page 68 for more information. (1065)
5. **Create directory script action:** The option to setup permissions of shares and the maximum number of connections for shares has been added. See *Script Action: Create Directory* on page 341 for more information. (1065)
6. **Create directory script action:** The option to set the owner of a directory has been added. The owner is specified by using the **Security** property of *Script Action: Create Directory* on page 341. (1065)
7. **Set User Group Memberships (AD) script action:** Property **Group names (Pre-W2K name)** has been added to allow the specification of multiple groups using variables. See *Script Action: Set User Group Memberships (AD)* on page 56 for more information. (1065)
8. **Setup user global group memberships script action:** By using the new option **Error if already member** you can configure the application not to generate an error when adding a user to a global group and the account is already a member. (1065)

9. **General:** To facilitate the specification of groups and other properties, you can now assign multiple values to a single variable. (1065)
10. **General:** The input data of all projects can now be exported and printed. (1065)

### Critical fixes

No critical fixes were found or reported

### Major fixes

1. **Active Directory:** The comma (,) character can now be used in the name of user accounts. (1033)
2. **Name generation:** When no name generation algorithm is specified, the application no longer uses the last 'configured' algorithm. This feature is particularly used when the user name is directly specified in the import file when creating user accounts. (1033)
3. **Name generation:** The names generated by the name generation algorithms can now be used in subsequent actions of the same script. Example: If you create a user account using a name generation algorithm you might want the algorithm to generate a separate name for an Exchange mailbox for the same user account. This name can now be stored in another variable that can be used in the action that creates the Exchange mailbox. (1033)
4. **Name generation:** The function **Add if empty** now correctly does not add the specified character if the name is not empty. (1065)
5. **Script execution:** The Security Identifier (SID) can now be exported in text format. (1065)

### Minor fixes

1. When a read-only project is started from the User Management Resource Administrator wizard, the project is now correctly shown in the projects bar (1033).
2. The on-line help is shown always when F1 is pressed. In previous versions, the on-line help was not activated when F1 was pressed and some windows were active (1033).
3. The help text of some of the sample projects has been updated (1033).
4. The information shown for the action Execute command line is no longer shown with the error icon. (1065)

### Build 1030, July 1, 2004

This is the first build of User Management Resource Administrator version 6.0.

## 2.2. Upgrading MASS projects from older versions

(Mass) Projects that have been created or used in the official release 7.5, **build 1263** or newer can be used directly in the current version. Mass projects that have been created in older builds than 1263 and not been used since, will need to be converted. Below text therefore only applies if you are upgrading from releases prior to build 1263.

### Upgrading mass projects from releases prior to version 7.5, build 1263

**The following text is taken directly from the help of version 1263.**

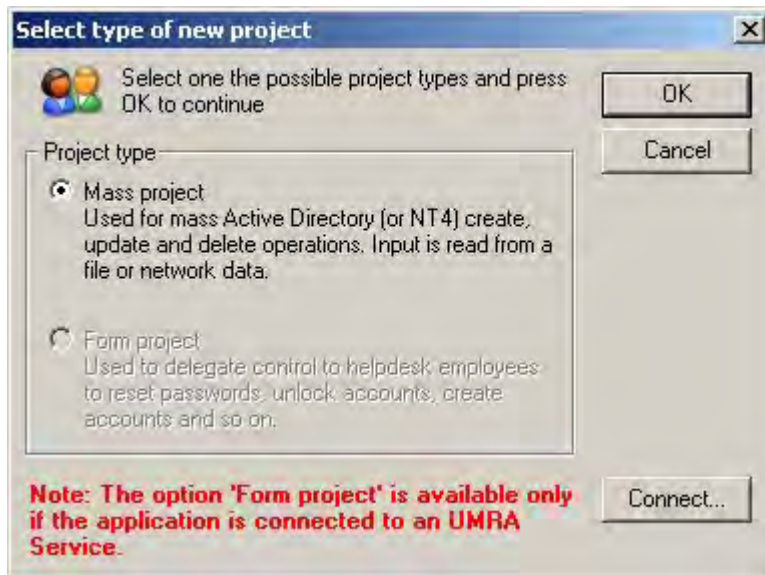
MASS projects created in previous versions of UMRA will not automatically work in the new version because the project type has changed in the new version. In the new version, there is only one UMRA project type. This project always includes a script which can take its input from various different sources, including a (CSV) file. This is also known as a MASS project.

### What has changed for MASS projects?

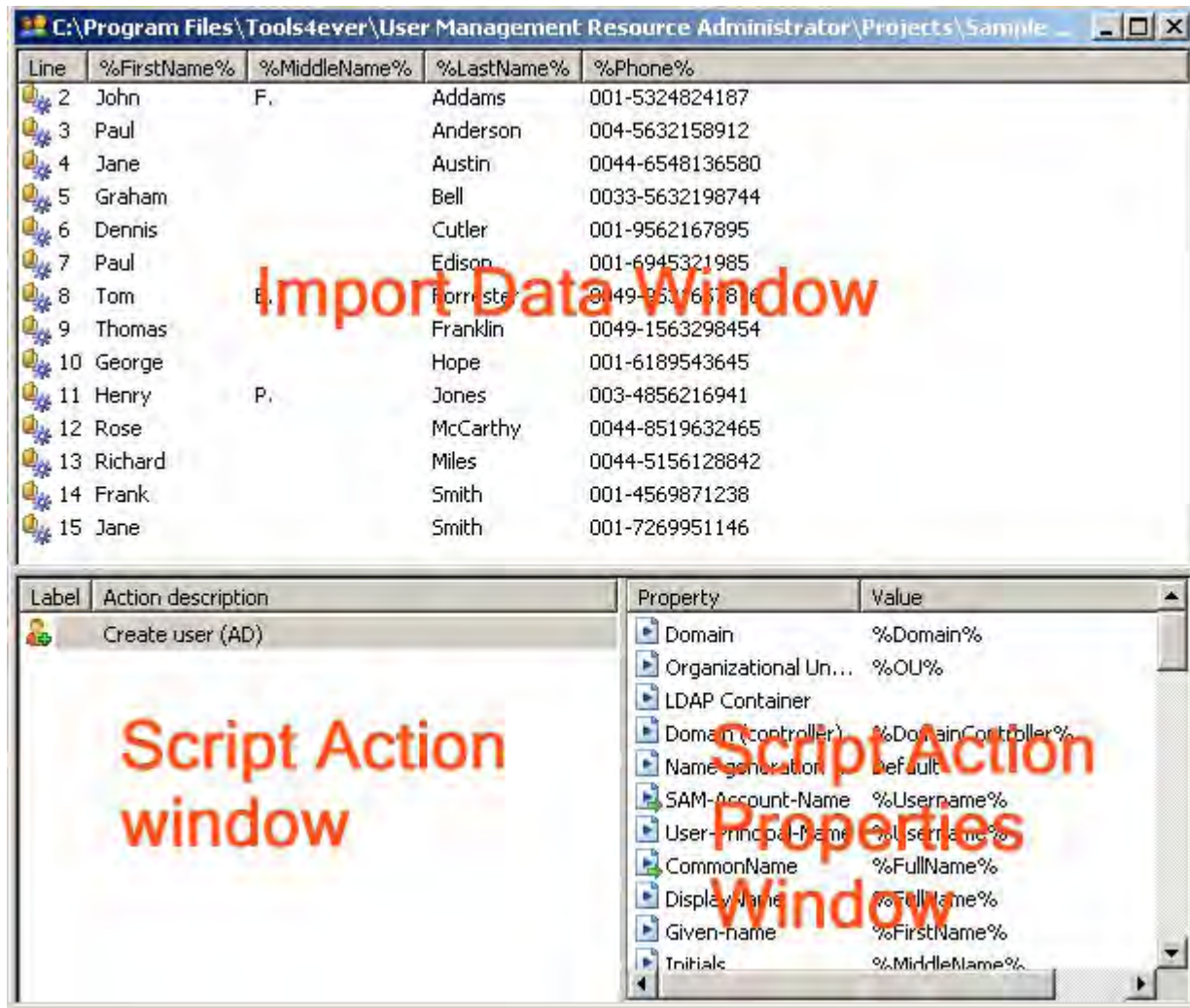
1. **Project type** - Because of the introduction of the new project type, the old MASS project files with the extension .upj are no longer used. Instead, all projects are stored in a workspace with the extension .uwp. The functionality of existing MASS projects will remain intact once these projects have been properly converted to the new format. See **Converting MASS projects created in previous UMRA versions** to find out how your existing MASS projects can be updated to the new format.
2. **Interface** - all functionality of the previous UMRA version has remained intact, but the location of certain functions and features has changed. An overview is given below.

### Main Interface changes

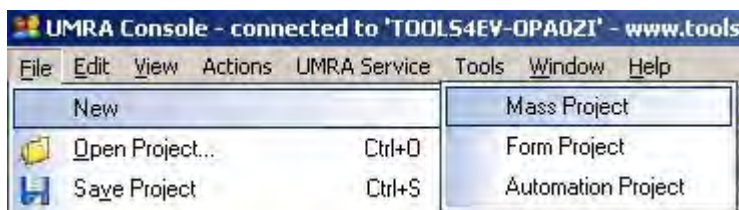
In PREVIOUS versions of UMRA, you could create a MASS project by selecting **File-->New** from the main menu and choosing the **Mass project** option:



In the next screen, file data could be imported in the top window, script actions could be added in the lower left window and the script action properties were specified in the lower right window (see screenshot below).



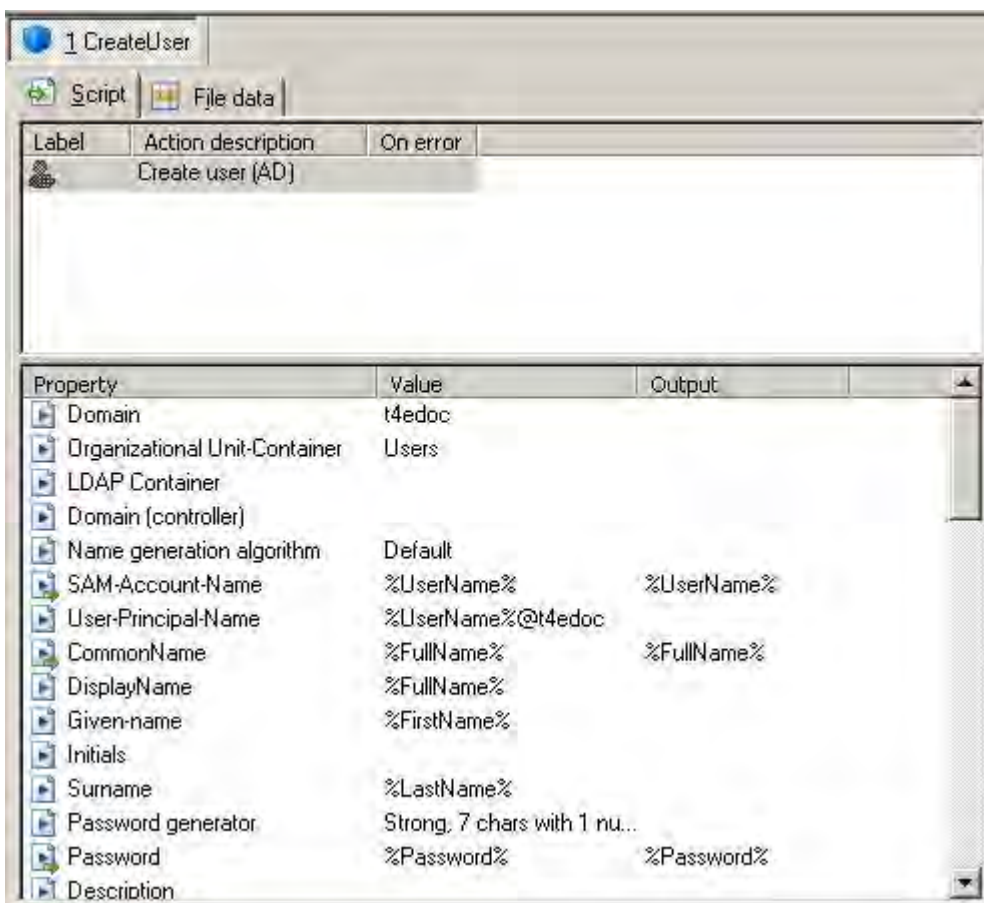
In the NEW UMRA version, there is only one project type. A new MASS project is created by selecting **File-->New-->Mass project** from the main menu.



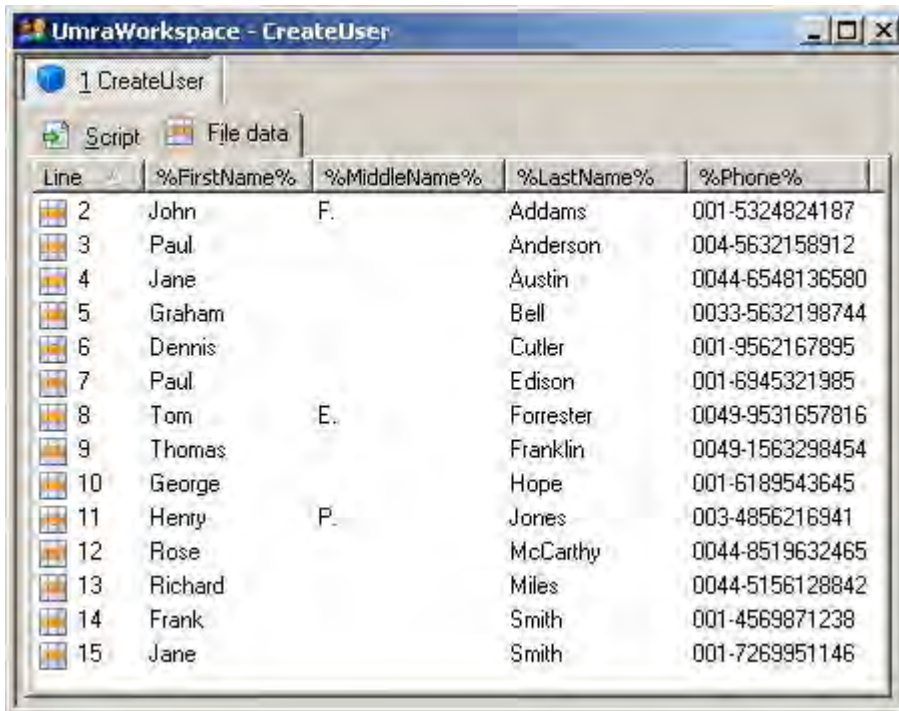
UMRA will automatically create a workspace for the new project. This workspace can be regarded as a collection of one or more projects. The new workspace contains the new project and some project components which you will need to specify. This includes a **project script** component and a **file data** component.



In the **Script component**, a project script can be created by adding script actions and specifying script action properties.



In the **File data** component, the file data to be used as input for the project script are defined.



Line	%FirstName%	%MiddleName%	%LastName%	%Phone%
2	John	F.	Addams	001-5324824187
3	Paul		Anderson	004-5632158912
4	Jane		Austin	0044-6548136580
5	Graham		Bell	0033-5632198744
6	Dennis		Cutler	001-9562167895
7	Paul		Edison	001-6945321985
8	Tom	E.	Forrester	0049-9531657816
9	Thomas		Franklin	0049-1563298454
10	George		Hope	001-6189543645
11	Henry	P.	Jones	003-4856216941
12	Rose		McCarthy	0044-8519632465
13	Richard		Miles	0044-5156128842
14	Frank		Smith	001-4569871238
15	Jane		Smith	001-7269951146

### Converting MASS projects created in previous UMRA versions

Please follow this procedure to update your MASS project:

1. Choose **File-->Open project** in the main menu.
2. Select your existing MASS project (with the extension .upj) in the list of projects under **Local UMRA project** and click **OK**.
3. The following message will appear: "The project file needs to be converted to the new UMRA format. Would you like to convert the project file now?". Click **Yes** to confirm, **No** to cancel the conversion. When you click **Yes**, a message will appear to confirm that the old file has been converted to a file with the extension .UWP (Umra Workspace Project).
4. The old file can no longer be used in the new version, so you are asked if you want to delete this old project file. Click **Yes** to confirm and click **No** to cancel.
5. **IMPORTANT** - If your MASS project contains references to other project files, these project files will have to be converted as well.



**Update references to project files in the script**

If your project script contains references to old project files (.UPJ), these names will have to be updated so that they correspond to the new project file name (the one which has been converted).

### **3. UMRA User Guide**



---

### 3.1. UMRA Basics

UMRA is a full blown enterprise solution for managing user accounts and all associated network resources. To benefit fully from UMRA's features, you need a basic understanding of the application's key concepts. These are briefly explained in this section.

The first section, **UMRA scripting**, explains the architecture of an UMRA script and how it is used to define a specific user or network management task. Such a task can be the creation of a user account for instance, together with a home directory, home share, group memberships, Exchange mailbox, etc.

In **UMRA projects**, it is explained how one and the same UMRA project can be configured to use input data from various different sources.

**UMRA Components** provides an overview of the various software components that together make up the UMRA solution.

Finally, UMRA project management provides a brief definition of UMRA projects and workspaces.



*Read the full PDF version of UMRA Basics*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/umra-basics.pdf>

#### 3.1.1. Introduction

UMRA is a full blown enterprise solution for managing user accounts and all associated network resources. To benefit fully from UMRA's features, you need a basic understanding of the application's key concepts. These are briefly explained in this section.

The first section, **UMRA scripting**, explains the architecture of an UMRA script and how it is used to define a specific user or network management task. Such a task can be the creation of a user account for instance, together with a home directory, home share, group memberships, Exchange mailbox, etc.

In **UMRA projects**, it is explained how one and the same UMRA project can be configured to use input data from various different sources.

**UMRA Components** provides an overview of the various software components that together make up the UMRA solution.

Finally, UMRA project management provides a brief definition of UMRA projects and workspaces.



*Read the full PDF version of UMRA Basics*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/umra-basics.pdf>

### 3.1.2. UMRA scripting

#### Scripts, actions and properties

##### UMRA Script

An UMRA script is used to perform a specific user account or network management task. This can be the creation of a user account for instance, together with a home directory, home share, group memberships, etc. for a new user.

##### Script actions

An UMRA script is made up of one or more script actions. An UMRA script for the creation of new users for instance, may include a script action to create a user account in Active Directory and a second script action to create a home directory for the new user. All the script actions together define the project script.

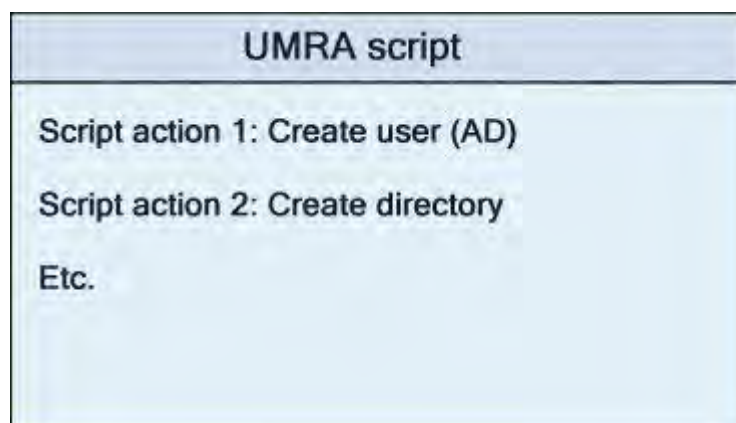


Figure 1: An UMRA script contains one or more script actions

The available script actions are fixed, which means that you cannot create your own script actions. Most script actions are however highly configurable, and some are very general, so this hardly imposes a limit. UMRA comes with a wide range of script actions, covering virtually every aspect of user account and directory management.



Figure 2: Sample of available script actions in UMRA

Apart from these standard script actions, there are also script actions available to perform some basic programming tasks, like evaluating a certain condition.

Script actions are added to the script by dragging a script action object from the list of actions and dropping it in the script pane of the project.

#### Script action properties

The script action properties of a script action provide information which specify in detail how a script action should be executed. In case of a

**Create User** script action for instance, the following properties are included:

- **Name of the domain** where the user account should be created
- **Name of the Organizational Unit** to which the user belongs
- **Common name**, which is the full name of the user account.

This concept is shown in the figure below. The user's common name has been specified here as "John Williams" in the **CN** script action property. This user should be created in the domain "Tools4ever.com", which is specified by the **domain** script action property, in the organizational unit "Sales\USA" as specified by the **OU** script action property.

For each script action, the script action properties that need to be specified are different. If we take a look at another script action, **Create directory**, the properties include the following information:

- Share name
- Name of the user's home directory

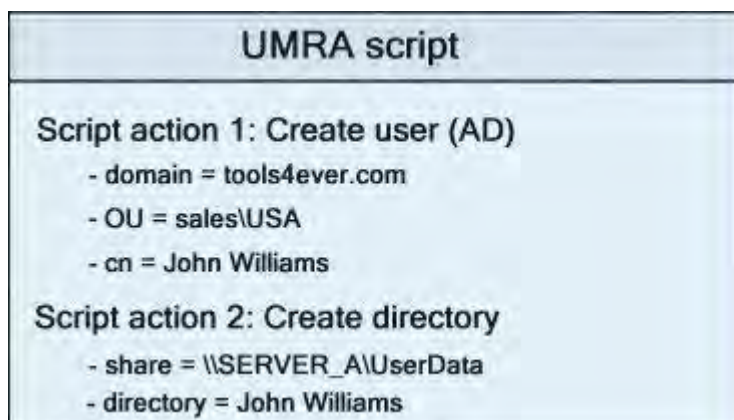


Figure 3: Specifying how a script action should be executed

### Variables

Let's return to the example shown in the previous Figure. You will note that the script action properties for these two script actions are all specified as fixed values. This script action will only be able to create the user "John Williams" in the OU "sales\USA" of the "tools4ever.com"



domain. But what if you want to create another user or if you want to specify another OU or domain?

To make a script generic, UMRA makes use of variables instead of hard set values such as “John Williams” or “sales\USA”. These variables represent a real value. This is shown in figure 4.

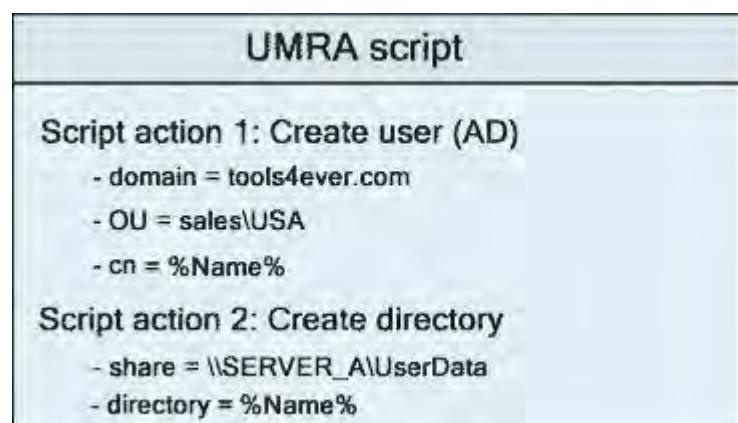


Figure 4: Specifying script action property values using variables

The name of the user ‘John Williams’ is now replaced with the variable **%Name%**. The name of the variable is not really relevant since it is only used as a placeholder to store the real value. When the script is executed, the variables will be replaced with their actual values and the network will be updated accordingly. In UMRA, the name of a variable is always placed between percentage signs:

- %Name%
- %Domain%
- %HomeServer%
- Etc.

The actual value of a variable is obtained from the input data to which the variable is linked. These input data may come from a (CSV) file, information provided by the end user, a database, etc. This is explained in more detail in *UMRA Projects* on page 9.

### 3.1.3. UMRA Projects

Each and every UMRA project contains at least a project script, which is the project's core component.

There are two key areas that determine the nature and possibilities of an umra project:

1. **Input Specification:** Where and how does a script get the information required for its operation?
2. **Execution control:** Where and when should the project and script be run to perform its operations?

#### **Input Specification.**

A script consist of a sequence of script actions. What exactly a particular script action does, depends completely on the contents of the particular script action's properties at run time. These in their turn are completely determined by the value of the variables that the script uses. There are many ways to specify the contents of these variables. The major methods are listed here.

#### **Information is directly specified in the script**

Some of the required information may be explicitly provided in the script itself when the script is run. For instance a script may use the script actions "set variable" to explicitly set a value for a certain variable.

#### **Information is the result of an other script action**

Many script actions query the network for certain information. The resulting information is often stored in variables that are used by other script actions further in the script.

#### **Information is gathered from external sources before the script is executed**

The required information may also (additionally) be gathered by the project from external sources before the script is executed. These options are:

File input data – the information the script needs to work with comes from a (CSV) file. The project will repeatedly execute its script, once with the information found in each (selected) line of the file.

Network input data - the information the script needs to work with comes from a network call, the script is executed once for each (selected) object found in a network query.

Form input data – the information the script needs to work with is entered or selected in a simple form. The form is created by the administrator using UMRA and delegated to a non-admin (e.g. Helpdesk employees). This category is therefore also known as Forms & Delegation..

Application input data – the information for the project script is provided by another information system or program. An example of this is the command line interface of UMRA, where the values entered on the command line specify the value of the variables used by the script.

In the following sections these categories will be described in more detail.

### **Execution Control**

There are several options as to **where** and **when** a project and script can perform its operations, the major methods are listed here.

#### **Directly by the console**

The console application can be used as a stand-alone application that executes the projects in the security context of the administrative user that is currently logged on. In this case the projects are stored locally on the machine running the console. This is often used by a network administrator to perform mass modifications to the network, for instance in order to create new user accounts and associated data from information available in a .csv file. This option can be used with either file input or network input data.

The software module that runs projects in this fashion is for historical reasons often referred to as the "mass module", although the term "console" module probably is a better description since much of the

mass features have since been ported to other parts of the software. Whenever the term "mass module" is used, it refers to projects directly run by the console, and NOT to mass-like features in other parts of the software.

**By the scheduler of the UMRA service**

Projects can be executed at scheduled times by the Umra service. The console application is used to configure the service, and to create the umra projects, that are stored and maintained by the UMRA service. All script actions run in the security context of the UMRA service.

This option can optionally be used with either File Input Data or Network Input Data.

**By the UMRA service, initiated from the UMRA Forms client**

An end user can use the UMRA Forms client to access forms located on the UMRA service to execute their associated scripts and actions. All scripts and actions are run with the security context of the Umra service. Access control to the different forms is provided by security settings (ACL) in UMRA on the individual forms.

The scripts and the security settings are created and configured by an administrator by means of the UMRA console application.

**By the UMRA service, initiated from the "umracmd.exe" command line application**

From the windows command line, "umracmd.exe" can be run to execute the script of an project that is maintained on the server. Variables that are required by the script are read from the command line. The script runs in the security context of the Umra Service account. Access control to the different scripts is provided by the Umra service by security settings in UMRA on the individual scripts.

The scripts and the security settings are created and configured by an administrator by means of the UMRA console application.

### By the UMRA service, initiated from UMRA Com

Umra projects can also be executed programmatically by using the UMRA COM (Component Object Model) objects, that are distributed with UMRA. For instance the "umracmd.exe" application uses these objects to communicate with the Umra Service. These objects can be used in VB, VBSCRIPT, C++, ASP, and many other languages and environments to initiate the execution of UMRA Scripts. Often this is used to access UMRA functionality from within a web application.

### File input data

The input data may come from a (CSV) file which can be imported into UMRA. In the following figure, a CSV file is shown containing the users' first name, middle name, last name and phone number. The first line of the CSV file contains the column headers "FirstName", "MiddleName", "LastName" and "Phone". For each row, these columns hold different data.



Figure 5: File input data contained in a CSV file

When this file is imported into UMRA, these data are transformed into a table with the columns FirstName, MiddleName, LastName and Phone.

### Script execution

In an UMRA project that takes its input from a file, the script will be executed for each line.

It is easy to see that the script action property values should be different for each processed row. When your project script includes a **Create User** action for instance, the **Surname** script action property should have for example the value “Addams” for row 2, “Anderson” for row 3, etc. Hard entering the name “Addams” in the **Surname** script action property, would result in 14 users with the name “Addams”. The solution is therefore to link the relevant column (in this case the **Lastname** column) to a variable (e.g. %Surname%) which can then be used to specify the value of the script action property.

This concept is illustrated in the figure below. The **Surname** property value has been linked to the %LastName% column. UMRA will now obtain the **Surname** property values from the %LastName% column.

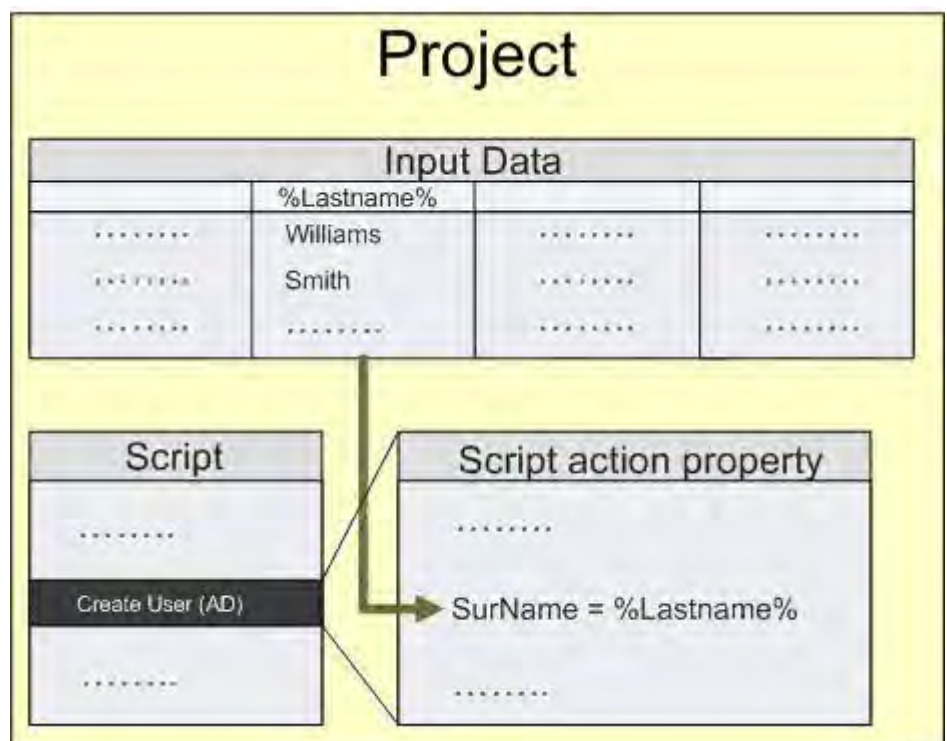


Figure 6: Linking input data columns to property values

When the project script is executed, the variables will be replaced with their actual value as found in the linked column. This is illustrated in the figures. The %LastName% variable has been linked to the second column

of the input data. When the first row is processed, the %LastName% variable is set to "Williams", for the second row it is set to "Smith", and so on.

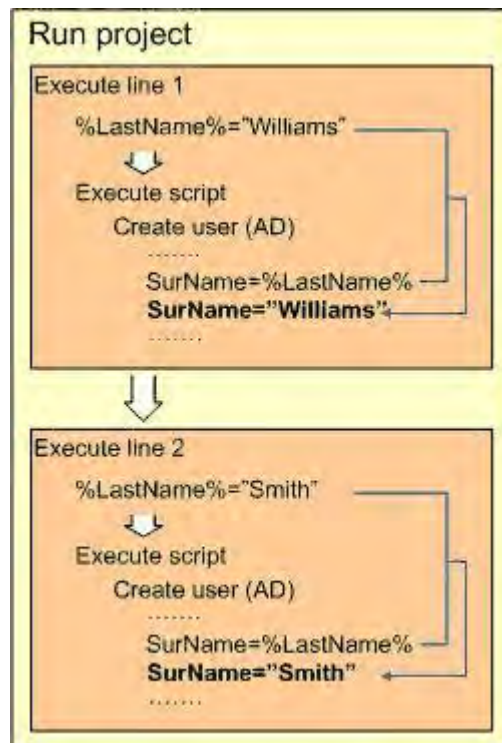
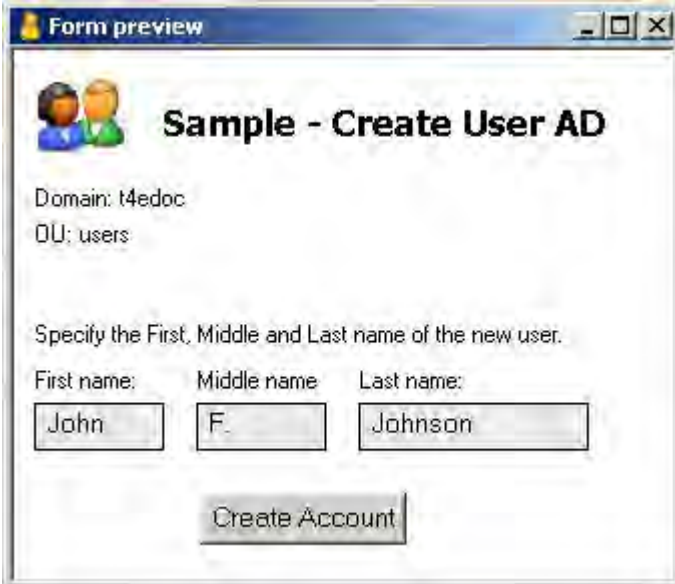



Figure 7: The project script is executed for each row of the input data

### Form input data

In this case, the UMRA project contains a script which takes its input from a form. Such projects can be used to delegate a specific user account management task, such as resetting a user's password or creating user accounts. An example of a simple form is shown in figure 8.



Form preview

 **Sample - Create User AD**

Domain: t4edoc  
OU: users

Specify the First, Middle and Last name of the new user.

First name: Middle name Last name:

John F. Johnson

Create Account

Figure 8: Example of a form to create new user accounts

By specifying some simple information (in this case the user's first, middle and last name) and clicking the action button **Create Account**, a user account is created using the specified name.

#### *Form layout*

To design the form layout, the administrator can use various different form fields. A form field is an object which is included in the form to describe the purpose of the form, to collect user information or to change the form's appeal:

- **Descriptive form fields**— the form field **Static text field** is used to explain the purpose of the form and the individual form fields to the end user. Examples: form title, input box description ("Please enter the domain name"), etc.
- **Interactive form fields** – the form fields **Input text box**, **Checkbox**, **Radio button** and **Table** are used to cater for interaction with the end user. Examples: Allowing the end user to enter the domain name in an input box, presenting a list of users in Active Directory which can be selected, etc.
- **Cosmetic form fields** - Pictures and vertical spaces can be used to make your form more appealing.



- **Action button** – When the end user clicks the action button, the entered or selected data are submitted.

Form preview

1

2

3

Domain: t4edoc

4

OU: users

5

6

Specify the First, Middle and Last name of the new user.

7, 8, 9

First name: Middle name Last name:

10 11 12

John F. Johnson

13

Create Account

Figure 9: Using form fields to design a form

- |               |                                  |
|---------------|----------------------------------|
| 1             | Picture                          |
| 2             | Static text field (title)        |
| 3 and 5       | Vertical spaces                  |
| 4, 6, 7, 8, 9 | Static text fields (description) |
| 10, 11, 12    | Input boxes                      |
| 13            | Action button                    |

*Script execution*

The interactive form fields (input box, radio buttons, checkbox, table column) of a form can be associated with a variable. These variables can be used by the project script. When the user clicks the action button, the project script will be executed, replacing the variables with the values entered or selected in the form.

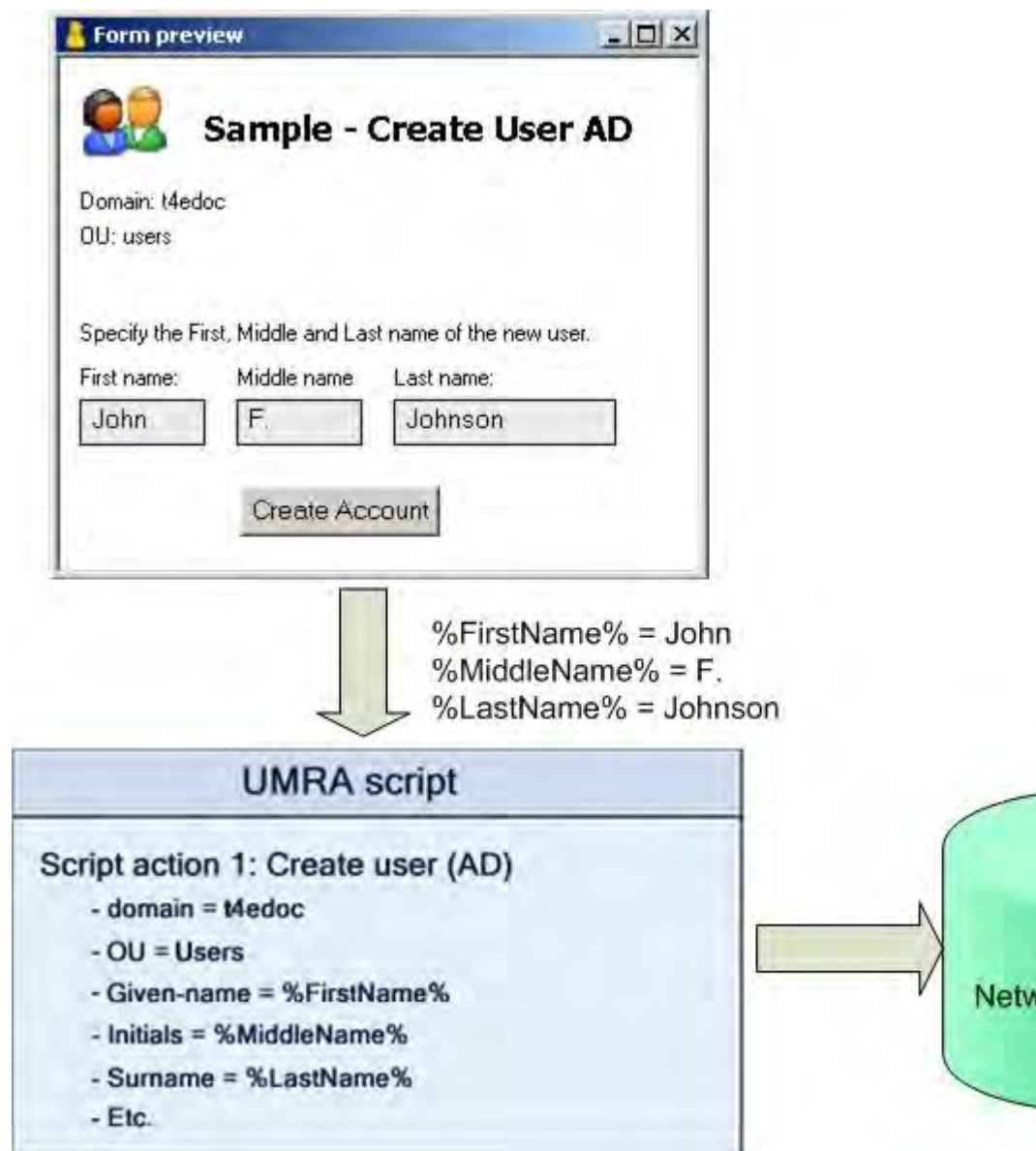


Figure 10 – Execution of a project script taking its input from a form

In case of the **Create User** example shown in figure 10, the information which the end user has entered in the form for the user's **first name**, **middle name** and **last name** is associated with the variables %FirstName%, %MiddleName% and %LastName%:

- %FirstName% = John
- %MiddleName% = F.
- %LastName% = Johnson

The project script includes the script action **Create User (AD)**. This script action holds, amongst others, the script action properties **Given-name**, **Initials** and **Surname**. The values for these script action properties have been specified using the variables %FirstName%, %MiddleName% and %LastName%.

As soon as the end user hits the action button **Create Account**, the specified data "John", "F" and "Johnson" are submitted and the project script is executed, substituting the variables %FirstName%, %MiddleName% and %LastName% with "John", "F." and "Kennedy" respectively. As a result, the network resource is updated.

#### **Application input data**

An UMRA project may also contain a script only which takes its input from other applications. This kind of integration, where other applications can use UMRA functionality, is achieved by using the Component Object Model (COM). This technology was developed by Microsoft to allow applications to interact with each other. The most important Microsoft applications that support COM are:

- Internet Information Services
- Office applications

All applications supporting COM use some kind of programming or script language to implement COM (ASP, VB(S), etc.). The procedure used is always the same:

1. The COM object is created;
2. The interface functions of the COM object are accessed;
3. Returned variables can be processed in the application.

An application can use multiple COM objects and COM objects can use other COM objects.

*UMRA COM*

User Management Resource Administrator contains several COM objects. With the UMRA COM objects registered, UMRA functions can be used within the ASP(X)/VB(A) script to execute an UMRA project and to read and process the project variables.

*Script execution*

The UMRA project script is executed by the UMRA server as instructed by the COM object. All variable names and values that are required to execute the UMRA project script, must be specified by Using the COM object. Usually, the values are taken from the application that uses the **UMRA COM** objects.

For more information about this project configuration, see the *UMRA COM User Guide*.

### 3.1.4. UMRA components

The UMRA solution contains several different software components.

These individual software components will be briefly described in this section.

#### **UMRA Console**

The **UMRA Console** is the main application for the administrator. The **UMRA Console** is used to build the actual user account or network management solution :

- develop and test UMRA project scripts. The UMRA Console comes with a wide range of script actions to build a project script and many testing facilities to test the script before it goes live in an operational environment;
- specify input data to be used in combination with the project script;
- set the security settings which specify who is allowed to execute the project script and form;
- structure UMRA projects;
- setup the **UMRA Service** (explained in the next section);

Projects can be developed to be run by the UMRA service, or local projects can be developed to be directly run by the console.

Projects that require forms, scheduling or are initiated by means of the UMRA COM, can only be executed by the UMRA service, but are developed using the UMRA Console application.

#### **UMRA Service**

Except for local UMRA projects that are executed directly by the UMRA console, all UMRA projects are stored on the **UMRA Service**.

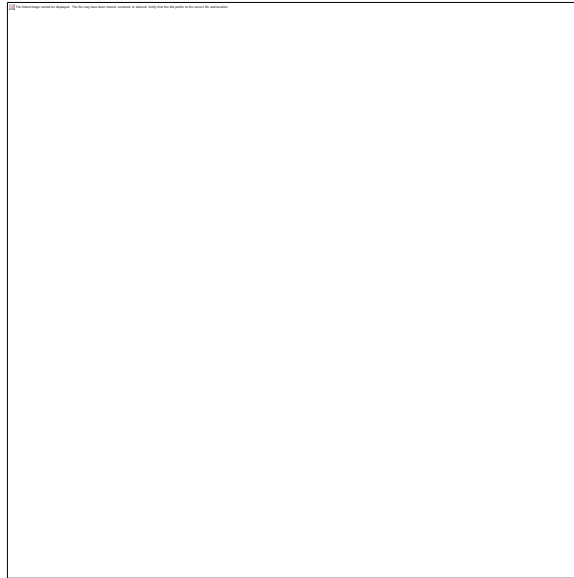


Figure 10: UMRA projects maintained by the UMRA service

The **UMRA Service** performs the following tasks:

- Maintains the UMRA projects
- Verifying access privileges of the currently logged-in user to check if the user is allowed to execute the project script;
- Executing the project script.
- contains a scheduler for the scheduled execution of project scripts
- Interfaces with the UMRA COM component so that external applications can initiate the execution of project scripts

#### **UMRA Forms client**

The **UMRA Forms** client is a windows application that is used to access and fill in forms maintained by the UMRA service.

This client is used by non-admins to connect to the **UMRA Service**. It presents the user with a list of names of those UMRA projects he is entitled to use. When a project is selected, the form of the project is presented to the user. The user can fill in the form and send it to the

service, which executes the project script, using the specified information in the form. If the project requires more information, it can present additional forms. It can also present additional forms to report information to the user.

**UMRA COM**

UMRA COM is a software component which is used to integrate UMRA functionality with other applications such as IIS (SharePoint). UMRA COM holds a variety of functions that can be used within a script (e.g. ASP(X)/VB(A) ) to execute an UMRA project and to manage the UMRA project variables.

**UMRA command line**

This software component is a command line interface that uses the UMRA COM object to execute an UMRA project maintained by the UMRA service.

This makes it possible to start a specific umra project from the Windows command line

### 3.1.5. UMRA project management

A user or network account management solution in UMRA can become very complex. To help you to structure your solution, UMRA offers a very flexible project management environment.

#### UMRA projects

In its simplest form, an UMRA project contains a project script only. Depending on the kind of solution you require, additional information can be added which should be used by the project script:

- **file input data** – user information contained in a (CSV) file can be imported into UMRA. During the import, these data are transformed into a table. The columns can be linked to a variable and used in the project script. This is covered in detail in the section *File input data* on page 12 of chapter *UMRA Projects* on page 9.
- **network input data** – instead of providing user information in a file, you can also make a network selection.
- **form** - The form is the template which the administrator creates for the non-admin to perform a specific user account management task. The data specified by the end user can be linked to variables and used in the project script. This is discussed in detail in the section *Form input data* on page 14 of chapter *UMRA Projects* on page 9.
- **security settings** - for all projects which are maintained by the **UMRA Service**, you need to specify who is allowed to execute the project form and / or project script.

For UMRA, there can be three different kinds of user accounts:

User account with	Description
Full control	These user have access to push forms to the UMRA service, setup, delete, manage all forms, project scripts and security settings. The number of user accounts with this type of access should obviously be very limited.
Form access only	These users can see and submit a form. When such a user connects to the <b>UMRA service</b> using the <b>UMRA Forms</b> client, the form is presented to them. The user can then specify the various fields of the form and let the UMRA service execute the script of the form project. The accounts can be configured for each individual form.
No access	These users can connect to the <b>UMRA service</b> but no projects will be shown.



### UMRA workspaces

Projects are contained in a workspace, which is a collection of one or more **UMRA projects**. By using a workspace it is easier to open all relevant projects at the same time.

### UMRA XML project and script files

UMRA can store UMRA projects and scripts in **XML** format. The XML files are human-readable and can for instance be used to copy (parts) of projects from one UMRA installation to another. The UMRA XML format is available with the following functions:

1. Main menu, File, **Import project (xml)**: The function allows you to import an UMRA XML project files. This project file can contain either a UMRA Service (Forms and Automation) of UMRA Console (Mass) project;
2. Main menu, File, **Export project (xml)**: Exports the current active project to an UMRA XML project file.
3. Project workspace, Script window, context sensitive menu: **Import script (xml and .usc)**: Import the UMRA script from the specified XML file. The script is imported at the current location of the UMRA script. The XML project file can contain either an UMRA XML project or UMRA XML script.
4. Project workspace, Script window, context sensitive menu: **Export script actions (xml)**: Export the current selection of UMRA script actions to an UMRA XML script file. The result file can be used to import the script actions in another UMRA script.
5. UMRA Service, Manage service projects, **Import**: Imports one or more UMRA XML project files.
6. UMRA Service, Manage service projects, **Export**: Exports the selected UMRA service projects to XML files.
7. UMRA Service, Manage service projects, **Backup**: Creates a directory with a name containing the date-time and UMRA build number, and stores all UMRA Service projects as UMRA xml project files in the directory

---

## 3.2. Getting Started

UMRA has been developed as a broad solution for managing Active Directory and other directory services. It covers the following user management areas:

- **Mass updating network resources** - UMRA offers administrators a fast and reliable solution for implementing bulk changes in network and associated resources. UMRA should be used for small to large migration projects and is an alternative for manual actions or complex scripting. The user data UMRA needs to work with can be obtained in many different ways (e.g. as a result of a database query or a CSV file containing, for instance, the user's First Name, Last name and Middle Name). Using built-in script actions, you can mass create new user accounts in Active Directory, together with Exchange e-mail accounts, home directories, (random) passwords, dial-in permissions, Windows Terminal Server settings, global group memberships, and profile directories.
- **Delegation of user management tasks to non-admins** - UMRA offers administrators the option to delegate user account management tasks to non-administrative users within the organization (e.g. helpdesk employees). Administrators can define a delegation project consisting of secure forms to handle a specific user management task such as resetting passwords, creating and deleting user accounts, disabling accounts, moving accounts (cross-domain), etc. These forms can be delegated to non-administrative users. Using forms, it is also possible to delegate management of other resources such as Windows computer services and printer queues.
- **Linking network resources to other information systems** - UMRA offers the possibility of linking network resources to other information systems, such as a link between an HR system and Active Directory. In a scenario where an employee leaves the company for instance, the corresponding user account in Active Directory can be automatically disabled. This automation option provides unrivalled flexibility and eliminates the need for scripting solutions entirely.
- **Integration of network resources with (self service) web portals** - companies using a web portal can integrate the UMRA features with their intranet by simply adding web pages for specific user management tasks. Different options can be made available, depending on the user's role within the organization. When HR

employees log into the intranet for example, they can be offered a page to create new users in Active Directory. Similarly, an IT manager can be offered a list of disabled users and authorize the deletion of disabled user accounts.

This document will guide you through the various UMRA software modules covering these user management areas.

When you have finished this guide, you will be able to install UMRA and create a user management solution yourself.



*Read the full PDF version of UMRA Getting Started*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/umra-getting-started.pdf>

### **3.2.1. Introduction**

UMRA has been developed as a broad solution for managing Active Directory and other directory services. It covers the following user management areas:

- **Mass updating network resources** - UMRA offers administrators a fast and reliable solution for implementing bulk changes in network and associated resources. UMRA should be used for small to large migration projects and is an alternative for manual actions or complex scripting. The user data UMRA needs to work with can be obtained in many different ways (e.g. as a result of a database query or a CSV file containing, for instance, the user's First Name, Last name and Middle Name). Using built-in script actions, you can mass create new user accounts in Active Directory, together with Exchange e-mail accounts, home directories, (random) passwords, dial-in permissions, Windows Terminal Server settings, global group memberships, and profile directories.
- **Delegation of user management tasks to non-admins** - UMRA offers administrators the option to delegate user account management tasks to non-administrative users within the organization (e.g. helpdesk employees). Administrators can define a delegation project consisting of secure forms to handle a specific user management task such as resetting passwords, creating and deleting user accounts, disabling accounts, moving accounts (cross-domain), etc. These forms can be delegated to non-

administrative users. Using forms, it is also possible to delegate management of other resources such as Windows computer services and printer queues.

- **Linking network resources to other information systems - UMRA** offers the possibility of linking network resources to other information systems, such as a link between an HR system and Active Directory. In a scenario where an employee leaves the company for instance, the corresponding user account in Active Directory can be automatically disabled. This automation option provides unrivalled flexibility and eliminates the need for scripting solutions entirely.
- **Integration of network resources with (self service) web portals -** companies using a web portal can integrate the UMRA features with their intranet by simply adding web pages for specific user management tasks. Different options can be made available, depending on the user's role within the organization. When HR employees log into the intranet for example, they can be offered a page to create new users in Active Directory. Similarly, an IT manager can be offered a list of disabled users and authorize the deletion of disabled user accounts.

This document will guide you through the various UMRA software modules covering these user management areas.

When you have finished this guide, you will be able to install UMRA and create a user management solution yourself.



*Read the full PDF version of UMRA Getting Started*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/umra-getting-started.pdf>

### 3.2.2. Mass updating network resources - Mass module

#### Summary

UMRA offers a solution for implementing mass changes in a network resource (e.g. Active Directory). Based on a selection of input data, the administrator can assemble a project script using built-in script actions (e.g. **Create User**) which are executed for each line of the selected data and update Active Directory accordingly. There are dozens of script actions available, covering the whole spectrum of user management tasks. For a full list of script actions, see appendix A.

#### Working procedure

The administrator creates a project in UMRA which involves a few simple steps, as illustrated in Figure 1:

1. Providing input data
2. Building the project script by adding built-in script actions to execute specific user management tasks
3. Running the project script in test mode and check the results
4. Execute the project script to update Active Directory and all associated resources.

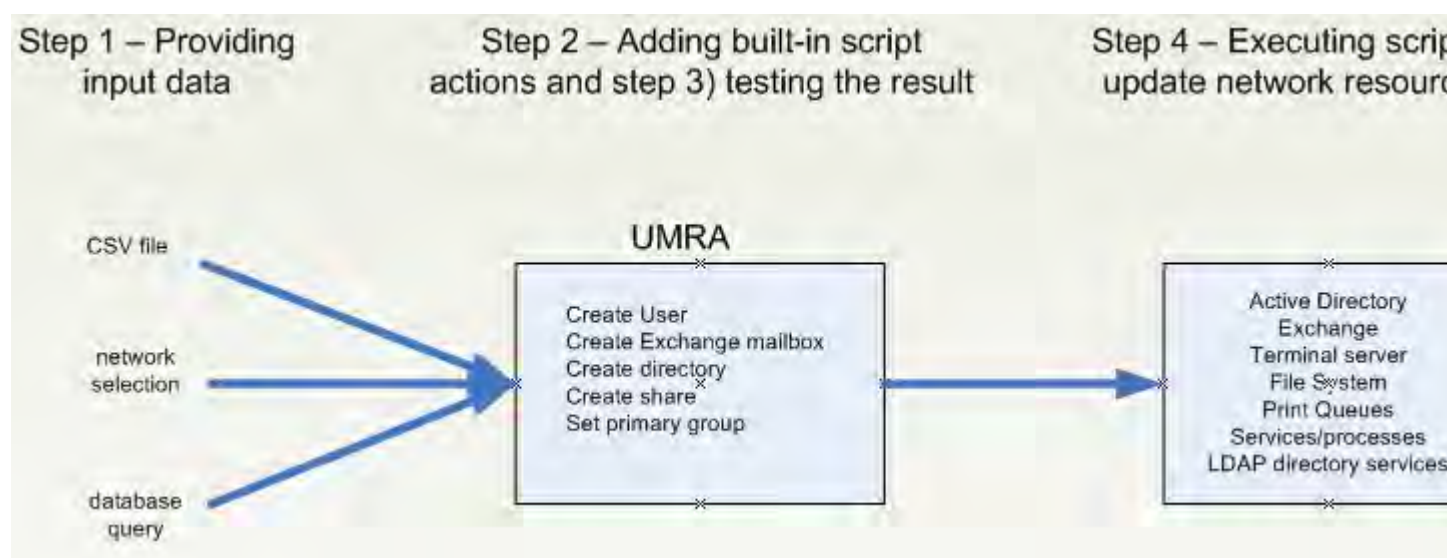


Figure 11: Implementing bulk changes in network resources

### Step 1 - Providing input data

The first step is to define the data UMRA needs to work with. In case of a mass create user solution, the input data are used to specify the user accounts that must be created. Each line of the input data will correspond with a single user account. An UMRA project takes its input data from either:

- a plain CSV file
- a network selection
- a database query

### Step 2 - Building the project script

In the next step, the administrator specifies the built-in script actions (e.g. **Create User**, **Move User**, **Set Group Membership**, etc.) which need to be executed for each line of the selected input data. No scripting knowledge is required. By simply dragging and dropping the built-in script actions in UMRA and setting the properties for these actions, the administrator can create a (complex) user management solution for mass updates:

Some examples:

- Bulk creation of user accounts (SAM, AD 2000 / AD 2003 and network resources)
- Mass update of a single attribute (e.g. telephone number)
- Migrating accounts from one domain to another
- Moving users across OUs and child domains
- Moving Exchange mailboxes to a new server
- Managing primary group memberships
- Creating and moving home directories and shares
- Etc.

### Step 3 - Testing the project script

You can execute your project script in test mode to check if it works correctly. In test mode, Active Directory and other network resources will not be updated. If errors occur when you execute the script, these will be displayed in the log window. After testing, you can switch to execution mode to update the network resource.

### Step 4 - Executing the project script

When the log window does not show any errors, you are ready to leave the test mode and execute the project script.

To become more familiar with the creation of such projects, you are invited to proceed to chapter *Creating a MASS example project - Mass create users* on page 12 which describes in detail how to create an UMRA solution for mass creating users.

### 3.2.3. Delegating user account management tasks - Forms module

#### Summary

UMRA also offers a solution which allows administrators to delegate (complex) user management tasks to non-admins using forms. Using UMRA to delegate user management tasks has some major advantages:

- Admin passwords do not have to be handed out to non-admins
- Administrators can be relieved from the burden of performing standard user management tasks (resetting passwords, moving users to another OU, setting group memberships, etc.). All these time consuming and repetitive tasks can be delegated to non-admins such as a Helpdesk employee.
- Very short learning curve for non-admin - The non-admins to whom the user account management task is delegated, is offered a very simple interface with which only the specified task (such as resetting a password) can be executed.
- For the administrator, the delegation procedure is simple and straightforward.

### 3.2.4.

#### Working procedure

The administrator creates a forms project in UMRA which involves a few simple steps:

1. Creating the form layout
2. Building the form script
3. Specifying security
4. Testing the script



### Step 1 - Creating the form layout

The first step is to create the form interface for the end user(s) to whom the user management task will be delegated (see figure 2).



Figure 12: Example of a delegation form. Using this form, created by the sysadmin, a non-admin is allowed to reset passwords.

*Figure - Example of a delegation form. Using this form, created by the sysadmin, a non-admin is allowed to reset passwords.*

To design a form, a wide variety of form fields is available. These form fields have various different functions:

- explain the purpose of the form to the user (static text, picture)
- let the user specify input data for the form project (e.g. input text for a password, table field to list users from Active Directory, etc.)
- initiate the execution of form actions - when the user clicks a button, the project script should be executed for the specified data (e.g. to reset the password of a selected user).
- make the form look more appealing (by adding a picture, vertical spaces, descriptive text, etc.)

### Step 2 - Building the form script

The script is at the heart of any UMRA project. In case of a delegation project, a project script needs to be executed when the user clicks a button in the form (e.g. a “Reset Password” button). The project script, which may consist of many different script actions, will then be executed using the input data

from the end user. In case of a **Reset Password** project for instance, the project script will obtain the user account of the selected user and set the password as specified by the end user. UMRA comes with a wealth of built-in script actions. For a full list of available script actions, see *Appendix A - Script actions* on page 64.

### **Step 3 - Specifying security**

The final step is to specify who will be allowed to access and run the form. Delegated users can the access these forms through the UMRA delegation client (UMRA Forms).

### **Step 4 - Testing the script**

The same testing procedure applies as for creating a MASS project.

To become more familiar with the creation of Forms & Delegation projects, you are invited to proceed to chapter *Creating a Forms example project - Reset password* on page 32 which describes in detail how to create an UMRA delegation solution for resetting a user's password.

### 3.2.5. User account provisioning - Automation module

This section provides some information regarding the two major application areas for the Automation module:

- Linking Active Directory (or any other LDAP directory service) to other information systems
- Integrating Active Directory in existing web portals

#### Linking Active Directory to other information systems

##### Summary

UMRA Automation is a solution which allows administrators to link Active Directory as well as other LDAP directory services with any other information system (SAP, PeopleSoft, Beaufort, etc.). Changes in the IS (an HR system for example) can be automatically propagated to Active Directory or other LDAP directory services.



Figure 13: Linking Active Directory (or any other LDAP directory service) with other information systems

##### Working procedure

1. **Identify the relationships between the changes in a(n) (HR) system and Active Directory (or other LDAP directory services)** - if a user is removed from an HR system for example, the corresponding changes to implement in Active Directory will have to be defined (e.g. removing the user object).
2. **Obtaining data from the information system** - this can be done in numerous ways. A file can be periodically read, a database query can be periodically executed or an (HR) system may perform some kind of active signaling.

3. **Executing UMRA project script to update Active Directory with data from the specified system.**  
If, for instance, an HR system processes the mutation of an employee leaving the company, this change could be automatically registered in a file and (periodically) read by UMRA. An UMRA project script can then be executed to update Active Directory accordingly by deleting the user account (or any other action).

## **Integrating Active Directory in existing (Sharepoint) web portals**

### **Summary**

Using UMRA Automation it is also possible to create a solution which integrates Active Directory with a company's intranet. Simple forms are made available to non-administrators (helpdesk or HR employees, for instance) using ASP/ASPX pages on an IIS website. The ASP(X) code of the Sharepoint / IIS website(s) integrates with UMRA using UMRA COM objects.

### **Working procedure**

1. Identifying the tasks to be supported in the Sharepoint / IIS portal.
2. Building the UMRA project scripts to execute the required tasks
3. Creating the Sharepoint / IIS website pages. This can be done with either ASP or ASPX pages. It is recommended to create these pages using the Visual Studio.Net development environment.

### 3.2.6. Installing UMRA

This section provides information about installing the UMRA application.

**Important:** the UMRA evaluation version can be installed on a local XP machine which is part of a domain. For evaluation purposes, there is no need to install UMRA or any DLLs on a domain controller of your network.

#### **Which module should I choose?**

When you install UMRA for the first time, you will be offered the following options:

**UMRA Console** - the **UMRA Console** is the Administrator's application for creating project scripts and forms.

**UMRA Forms** - **UMRA Forms** is the client application for the non-admins to access the form projects developed by the administrator in the **UMRA Console**.

**UMRA Automation** - This installs and registers several UMRA Com objects. This is needed for 3rd party programs to be able to issue commands to the UMRA software. This is for instance used in some scenarios of solutions that to link Active Directory with other information systems and to integrate Active Directory in existing company web portals.

Generally, when installing the console for the administrator, it is recommended to also install the other two modules.

When installing for an end user (e.g helpdesk employee) who is only going to perform pre-created form projects created by the administrator , only install the UMRA forms module.

When installing on a web server, in order to let the web server call Umra projects by COM, only install the Automation module.

#### **System requirements**

The following table shows all of the requirements to run the User Management Resource Administrator application.

Description	Required	Recommended
Operating system	Windows XP, Windows 2008 (all versions) Windows 2003 (all versions) , Windows 2000 (all versions)	
Supported network operating system	Windows 2008 (all versions), Windows 2003 (all modes), Windows 2000 (all modes), Windows NT4 (SP6)	
Required privileges of logged on user	Administrative access to Active Directory and/or all computers and domains with managed user accounts	
Available hard disk space	25 MB	40 MB or more
Processor	Pentium III, 600 MHz, AMD 900 MHz	Pentium IV, > 1 GHz or AMD > 1.6 GHz
System memory		512 MB or more

#### Installing the UMRA program files

**All of the User Management Resource Administrator software is contained in a single executable file: SETUPUSERMANAGEMENT.EXE.**

To install the UMRA application, please download the most recent UMRA build from [www.tools4ever.com](http://www.tools4ever.com) and follow the installation instructions. Once the files have been installed, the UMRA wizard is launched to guide you through the remaining UMRA installation.

In case your solution requires the installation of Exchange (e.g. for a project which includes the mass creation of mailboxes), please follow the procedure as described in Installing the Exchange system management tools for Exchange 2003.

#### Exchange 2000/2003 requirements

In order to use the Exchange 2003/2000 features within User Management Resource Administrator, you must have a functional Exchange server in your network. Additionally, it is required to have the **Exchange System Management tools** installed on the local machine that runs the **User Management Resource Administrator** application.

**Installing the Exchange system management tools for Exchange 2003**

1. Insert the CD containing the **Microsoft Exchange 2003** Software (standard or enterprise edition), and run **setup.exe**.
2. Under **Deployment**, select **Exchange deployment tools**.
3. Choose the option **Install Exchange System Management Tools Only**.
4. Follow the instructions for your specific operating system.

For **Exchange 2000**, the procedure is similar.

### 3.2.7. Creating a MASS example project - Mass create users

#### Project definition

In this section, we will create a solution to mass create new user accounts in Active Directory. The same steps will be followed as described in chapter *Mass updating network resources* on page 1:

- Step 1 - Providing input data
- Step 2 - Building the project script
- Step 3 - Testing the project script
- Step 4 - Executing the project script

This example can be extended with many additional script actions - your imagination is the only limit. A full list of available script actions in UMRA can be found in *Appendix A - Script actions* on page 64.

#### Step 1 - Importing your input data

For this example project, we will use input data from a CSV file. Each row of this file contains several fields with user information:

- first name
- middle name
- last name
- phone number

The fields in each row are separated by a comma delimiter. The first line of the CSV file (highlighted in the example shown below) contains the column headers. Here is an example of how such a file may look like:



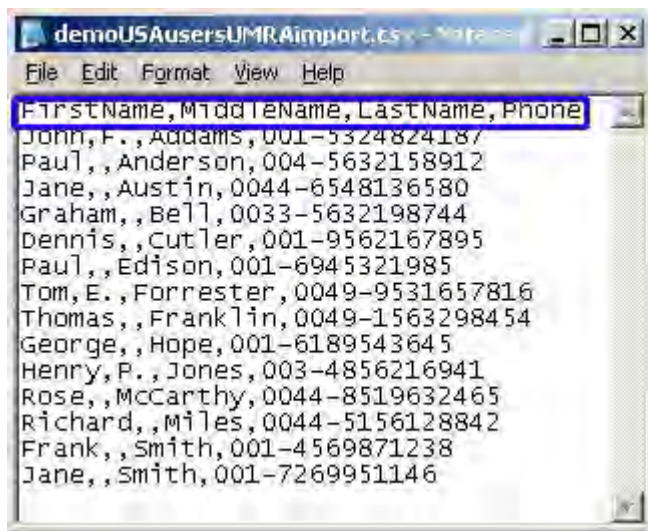


Figure 14: CSV file as input data source

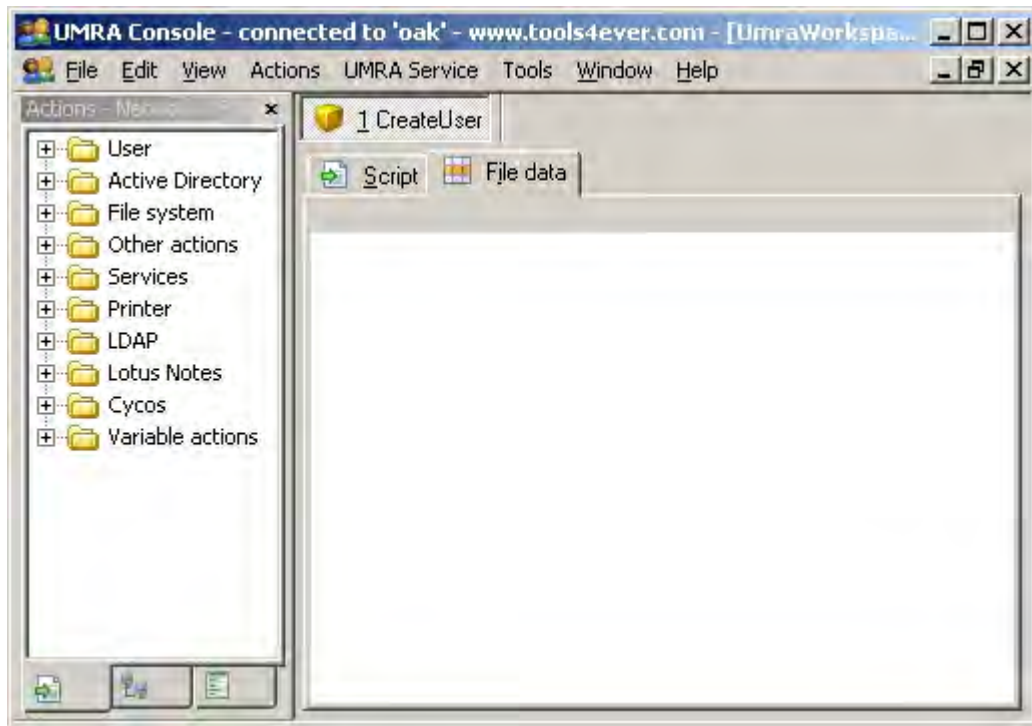
*Figure 1 - CSV file as input data source*

When the file is imported in UMRA, these data will be converted to a table where each row represents a user. Each column represents a field.

#### Importing a CSV file.

1. Create a .csv or .txt file like the one shown.
2. Start the UMRA Console and select the menu command **File,New,Mass Project**.  
The Program will ask for a new project name. This will open a new project with the specified name.  
The projects contains two tabs: A tab called "Script", and a tab called File.
3. Select the File data tab. At the moment the File data tab will still be empty

Equation 1: A new mass project



4.

Figure 2 - A new mass project

5. Right click in the file data pane and select **Import file data**, or select **File→Import**, and select the file you have just created. The following dialog will show:

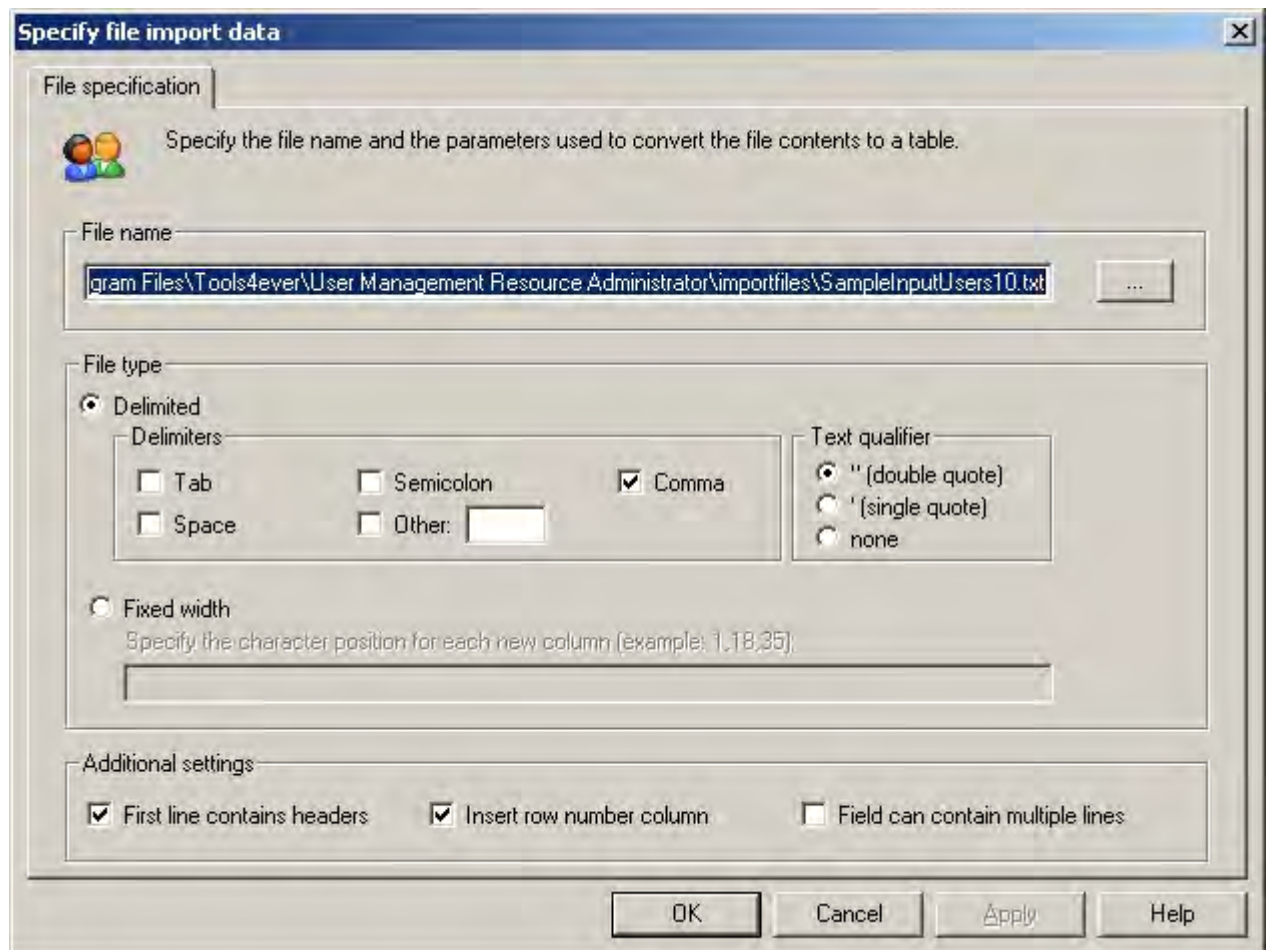
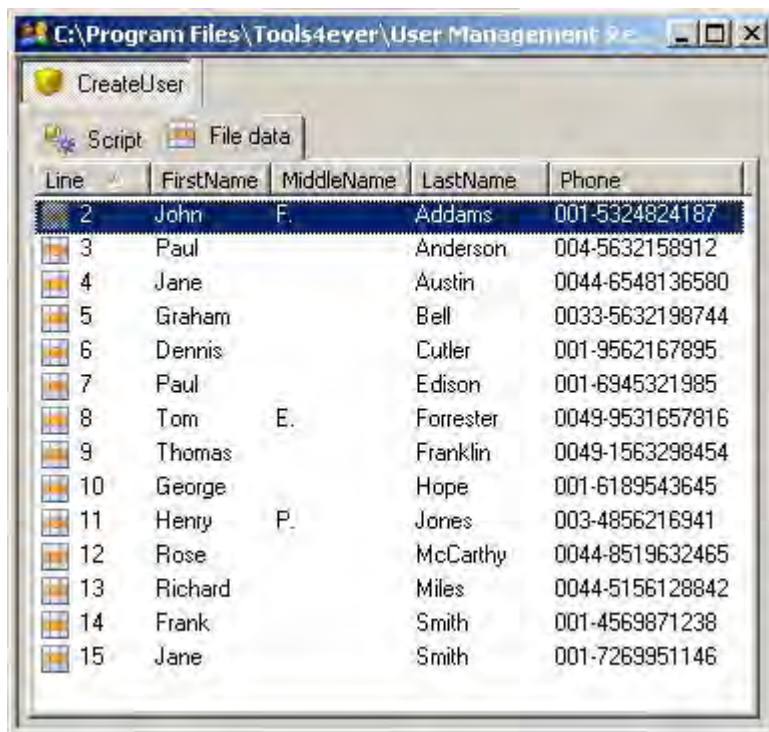


Figure 3 - Specifying the File Import data

6. Here you can specify how the CSV file should be read. In our case, the CSV column delimiter is a Comma, so you can select **Comma** under **Delimiters**. Under **Additional settings**, select the option **First line contains headers**. Click **Finish** to finalize the data import.
7. The imported data will be displayed in the **File data** window of your project as shown in figure 5:



Line	FirstName	MiddleName	LastName	Phone
2	John	F.	Addams	001-5324824187
3	Paul		Anderson	004-5632158912
4	Jane		Austin	0044-6548136580
5	Graham		Bell	0033-5632198744
6	Dennis		Cutler	001-9562167895
7	Paul		Edison	001-6945321985
8	Tom	E.	Forrester	0049-9531657816
9	Thomas		Franklin	0049-1563298454
10	George		Hope	001-6189543645
11	Henry	P.	Jones	003-4856216941
12	Rose		McCarthy	0044-8519632465
13	Richard		Miles	0044-5156128842
14	Frank		Smith	001-4569871238
15	Jane		Smith	001-7269951146

Figure 15: The data from the CSV file are converted to a table with a row for each line in the file and columns for each field

Figure 5 - The data from the CSV file are converted to a table with a row for each line in the file and columns for each field

## Step 2 - Building the project script

The next step is to create a project script which will be executed **for each row of the input data table**. A project script consists of one or more built-in script actions where each script action executes a specific task.

### Adding the Create User (AD) script action

1. Click the **Actions** tab in the **Actions-Network-Form fields** window. Open the **User→Active Directory** folder and drag the **Create User (AD)** action to the project script window. When fully configured, this script action will create a user account in Active Directory for each user listed in the input data.

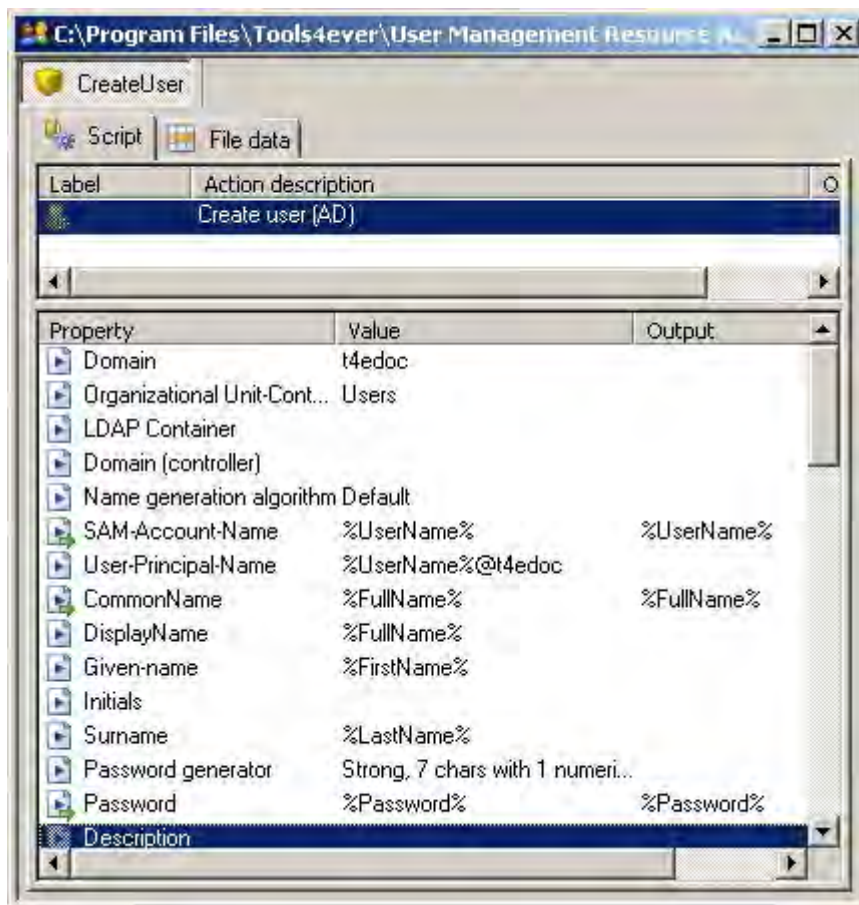


Figure 16: The Create user (AD) script action with its properties and property values

Figure 1- The Create user (AD) script action with its properties and property values

### Setting script action property values

Each script action in UMRA holds a different set of properties specifying in detail how the script action should be executed. In case of a **Create User** script action for example, it includes (amongst others) the following information:

- name of the domain where the user account should be created.
- name of the container where the user account should be created.
- User's logon name (SAM-Account-Name), which can be composed from the user's first and last name.

Some of these property values need to remain the same for each row of input data being processed. These property values, such as the domain name and the value for the container where the user account should be created, can be entered directly in the script action properties.



### Setting the Domain property value

1. Double click the **Domain** property of the **Create User (AD)** script action which you have just added to the project script. This will bring up the **Properties** window.
2. Enter the name of your domain in the **Use the following value** field. The domain name can either be entered using its DNS name (e.g. tools4ever.com) or its NETBIOS name (e.g. TOOLS4EVER).
3. Click **OK**.

### Setting the Organizational Unit-Container property value

1. Double click the **Organizational Unit-Container** property of the **Create User (AD)** script action. This will bring up the **Properties** window.

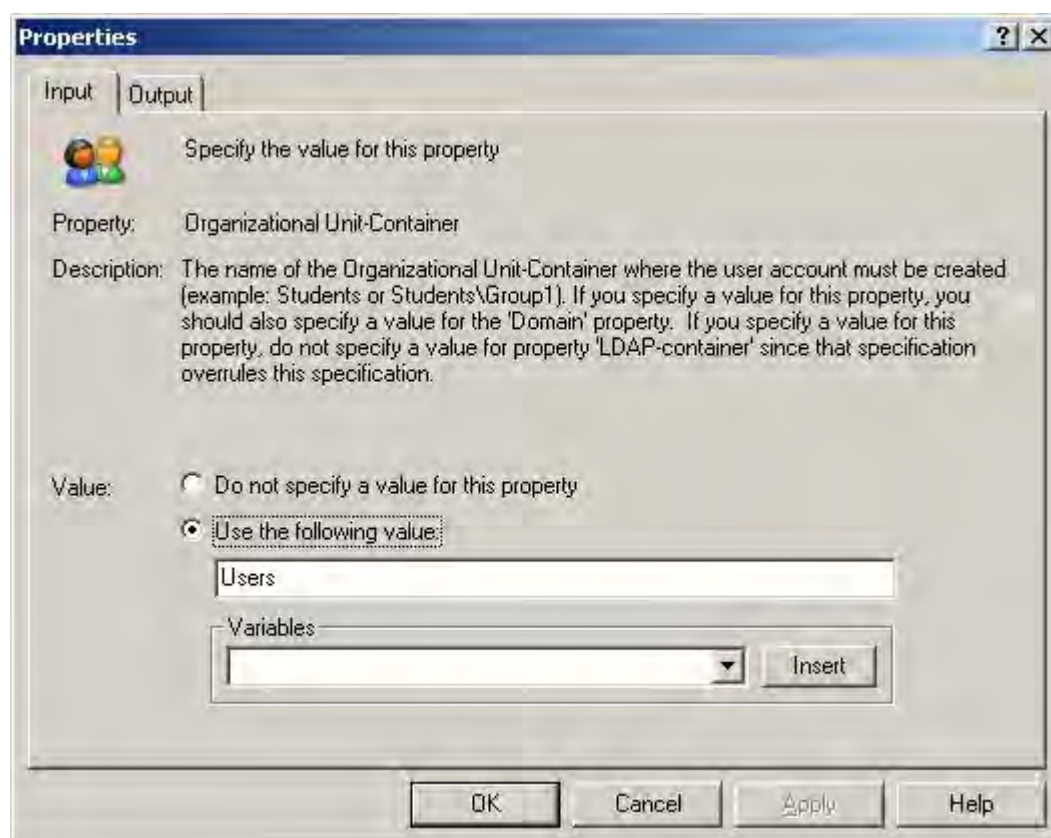


Figure 17: Specifying the OU in which the user accounts should be created

*Figure 2 - Specifying the OU in which the user accounts should be created*

2. Enter the name of the organizational unit container in which you want to create the new user accounts. By default, new user accounts are created in the container **Users**, but you can change this to any OU-container. To create new users in an existing OU container called "Marketing" for example, just enter **Marketing**.

### Setting the name generation algorithm property value

This algorithm can generate all user names found in Active Directory (Common Name, sAMAccountName, internet-style login name, etc.). These names can be generated in many different ways and completely in compliance with company naming conventions. For this example, the default algorithm will be used. Based on the user's first name, middle name and last name it will generate the various user names for Active Directory.

1. Double click the **Name generation algorithm** property of the **Create User (AD)** script action. This will bring up the **Properties** window.

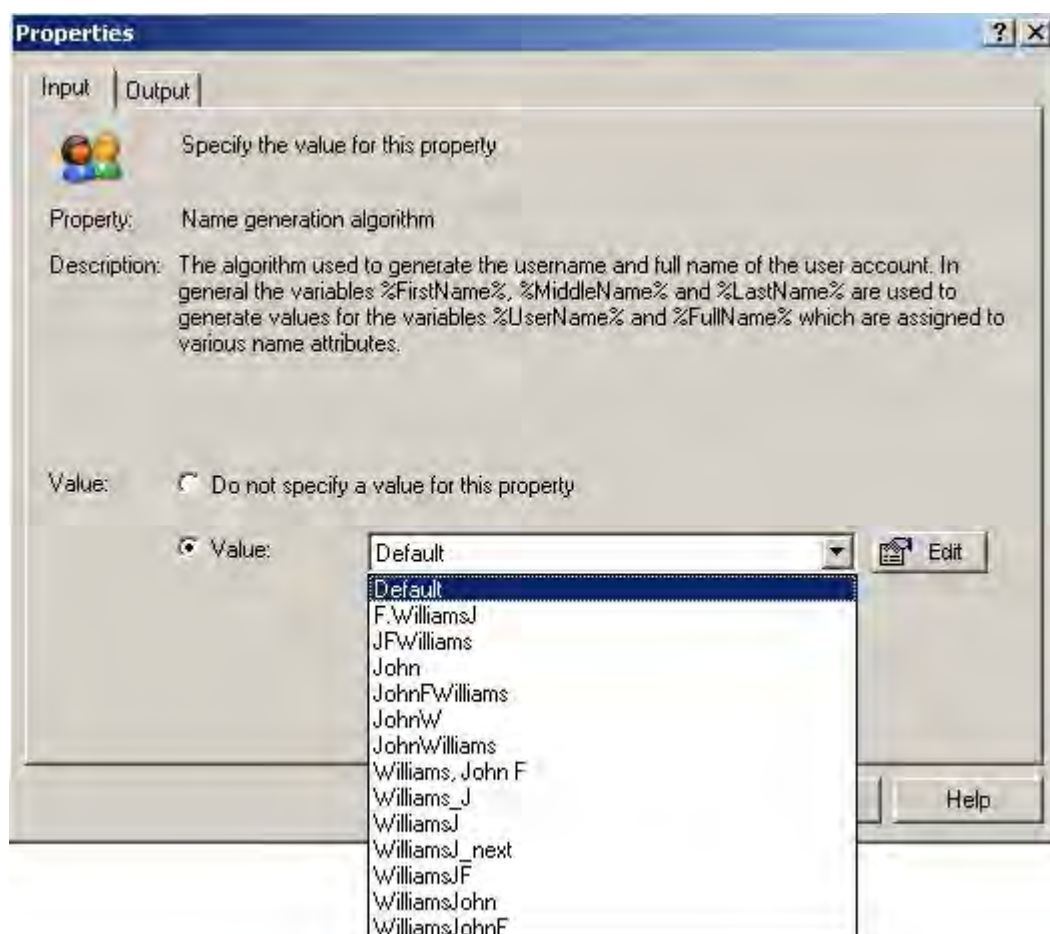


Figure 18: Setting the default name generation algorithm

*Figure 3- Setting the default name generation algorithm*

2. Choose the **Value** option. From the list of name generation options, choose the **Default** option.
3. Click **OK**.

### Setting the User-Principal-Name property value

This is an Internet style login name for the user. It is the preferred logon name for Active Directory users. Users should be using their UPNs to log on to the domain (using the syntax account\_name@domain.com). In UMRA, the account name is one of the user names generated by the name generation algorithm. For this example, only the domain name needs to be added.

1. Double click the **User-Principal-Name** property of the **Create User (AD)** script action. This will bring up the **Properties** window.
2. In the **Use the following value** field you will see the predefined syntax “%UserName%@%Domain%”. The variable %UserName% is automatically created by the name generation algorithm and does not have to be changed. Replace the %Domain% variable with the name of your domain (e.g. %Username%@tools4ever.com).
3. Click **OK**.

### Setting the Password generator property value

In UMRA, passwords can be generated automatically by the password generator. The generated password is stored in the %Password% variable and used to set the user's password in the **Password** property of the **Create User (AD)** action. If you want to create the same password for all users, you can specify the password directly in the **Password** property. For our example project, we will be using a predefined password generator setting.

1. Double click the **Password generator** property of the **Create User (AD)** script action. This will bring up the **Properties** window.
2. Select the option **Use the following value** and click the **Edit** button. This will bring up the **Password generator** window.



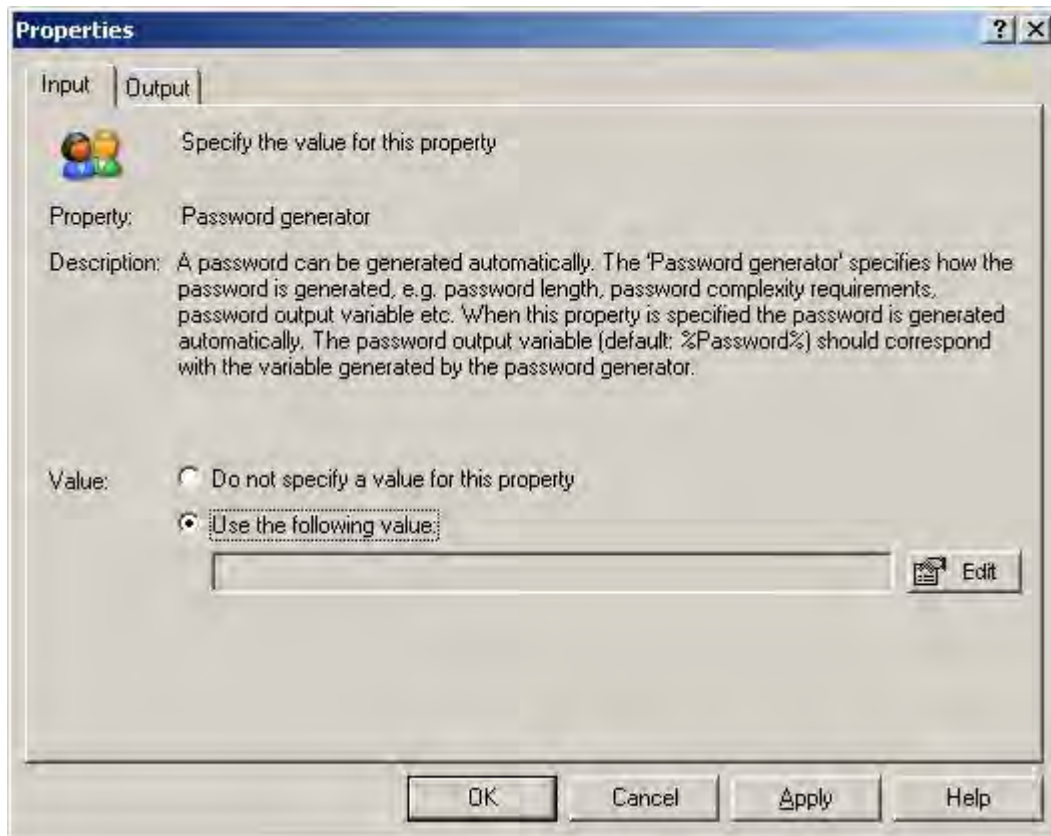
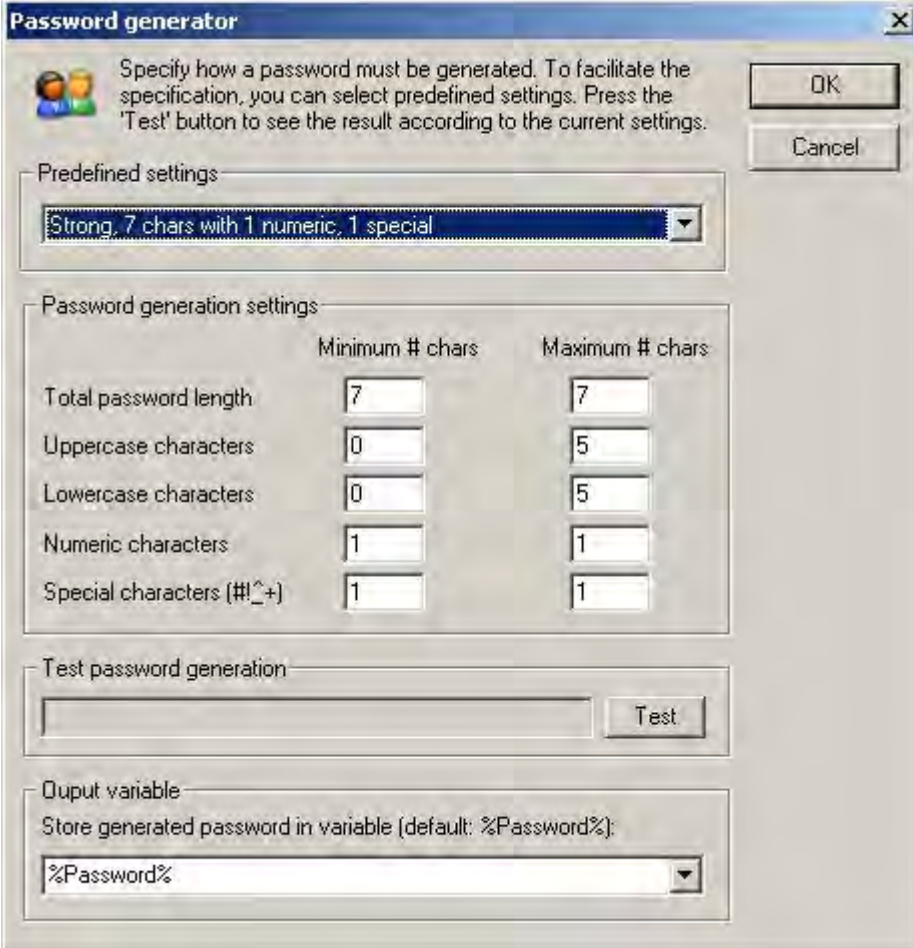


Figure 19: Password generator property

Figure 5- Password generator property

3. In the **Password generator** window, accept the default predefined setting ("Strong, 7 chars with 1 numeric, 1 special") by clicking **OK**.



**Password generator**

Specify how a password must be generated. To facilitate the specification, you can select predefined settings. Press the 'Test' button to see the result according to the current settings.

Predefined settings

Strong, 7 chars with 1 numeric, 1 special

Password generation settings

	Minimum # chars	Maximum # chars
Total password length	7	7
Uppercase characters	0	5
Lowercase characters	0	5
Numeric characters	1	1
Special characters (!@_+)	1	1

Test password generation

Test

Output variable

Store generated password in variable (default: %Password%):

%Password%

OK

Cancel

Figure 20: Specifying password generation options

*Figure 5- Specifying password generation options*

4. The selected predefined setting will now be displayed in the **Use the following value** field:

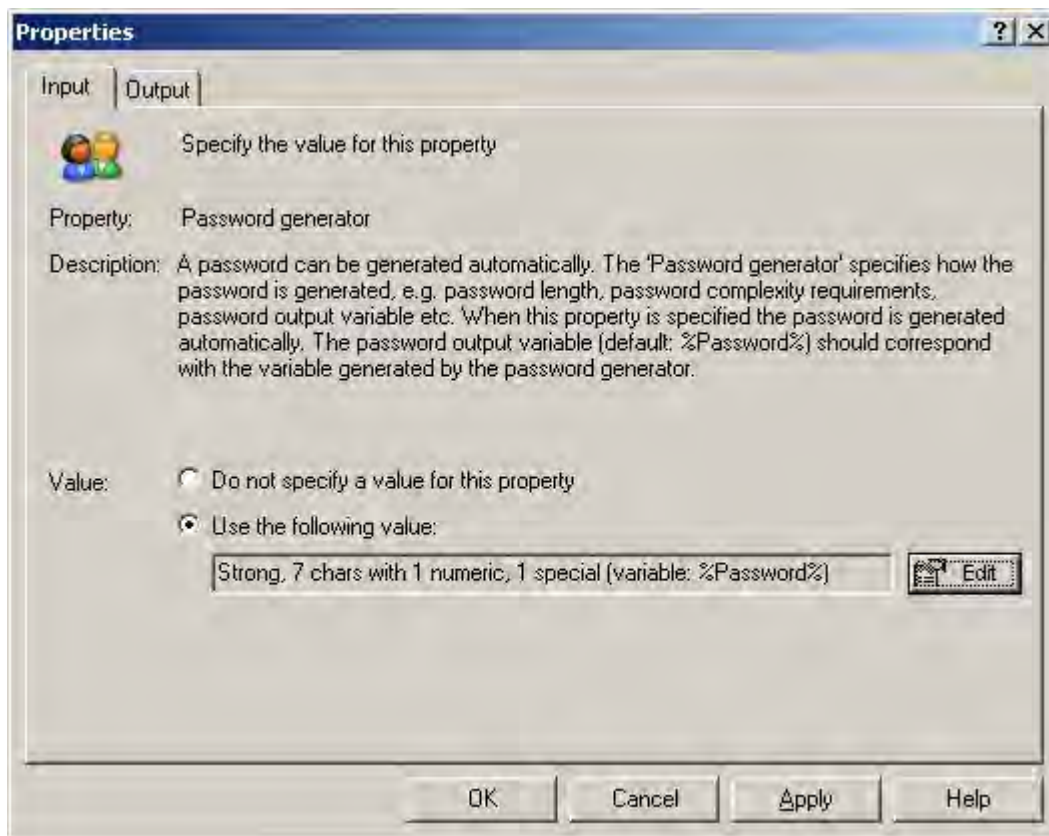


Figure 21: The specified password generation option as shown in the Properties window

*Figure 6- The specified password generation option as shown in the Properties window*

5. Click **OK**.

Alternatively, you can also specify in detail the rules used to generate a password. See the topic *Password generation* in the Online Help of User Management Resource Administrator for a detailed description.

### Setting the Telephone number-home property value

In this property, the home phone number of the user is set.

1. Doubleclick the **Telephone number-home** property of the **Create User (AD)** script action. This will bring up the **Properties** window.
2. Select the option **Use the following value** and enter **%Phone%**.
3. Click **OK**.

Note that this time, instead of entering a value directly, the name of a variable is entered. Why this is done and how this works, is explained in detail in the next section.

### Linking input data columns to variables

Unlike the property value for the domain name, most property values will be different for each row of input data. The **Surname** property for instance, may be “Williams” for the user in row 1, “Smith” for the user in row 2, etc.

For these properties, you cannot enter the actual value in the script action property. If you would do this for the **Surname** property for instance, each row of the input data would be processed using the same surname.

For this reason, a link can be established between the input data columns and the property values. This is done by assigning the input data columns to a placeholder, also called a variable (e.g. **%LastName%** for the **LastName** column). These variables can then be used in the script action properties to represent the property values.

In figure 14, the **Surname** property value has been linked to the **%LastName%** column. UMRA will now obtain the **Surname** property values from the **%LastName%** column.

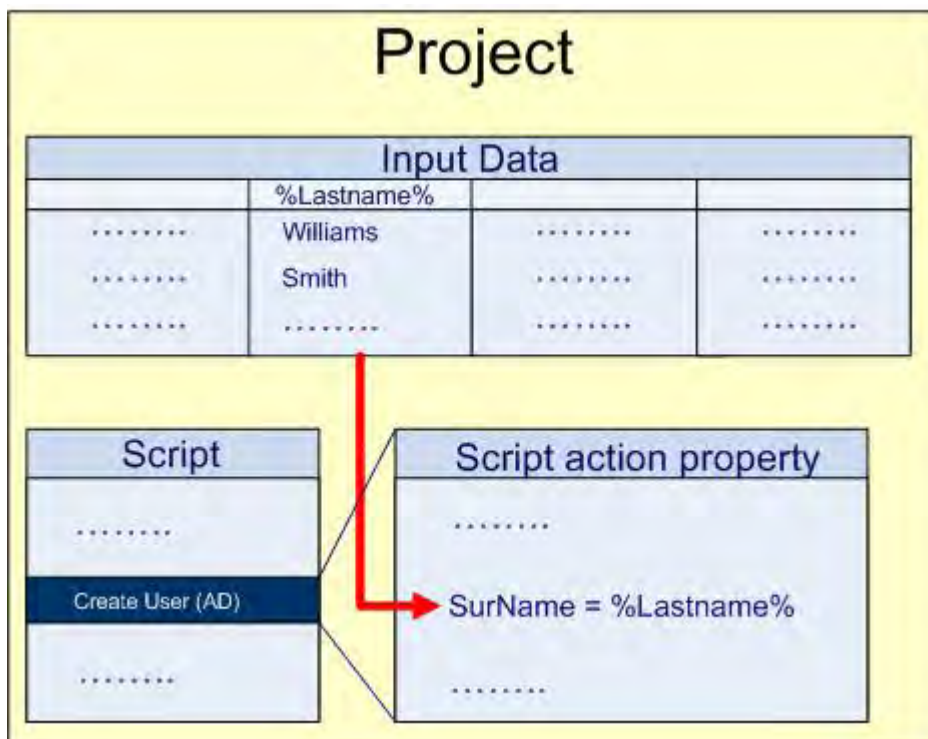


Figure 22: Linking input data columns to property values

When the project script is executed, the variables will be replaced with their actual value as found in the linked column. This is illustrated in figure 15 where the **Surname** property value has been linked to the **%LastName%** column. When the first row is processed, the **%LastName%** variable is set to “Williams”, for the second row it is set to “Smith”, and so on.

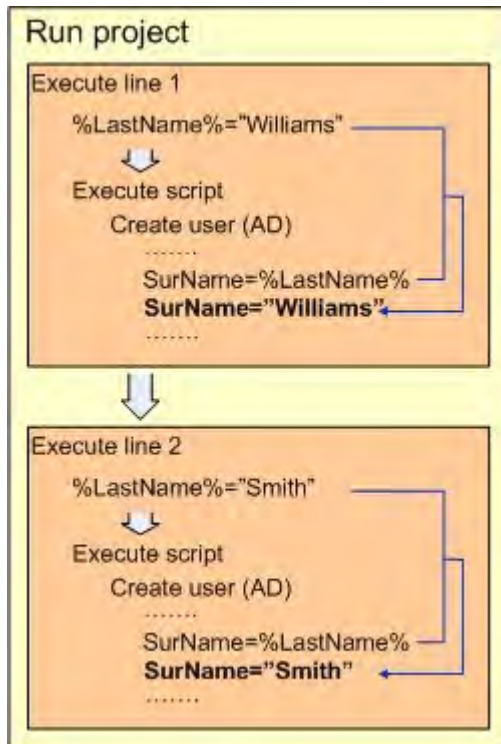


Figure 23: Specifying variables for property values

Many script actions have predefined variables for the property values. By default for instance, the **Create User (AD)** script action uses the **%FirstName%** variable to represent the property value for **Given-name**.

When you right-click an input data column, these predefined variables are automatically displayed. When you select a variable, it is automatically linked to the selected column.

All other variables have to be manually defined.

In the final steps, we will put this theory into practice.

1. Right-click the **FirstName** column and select the **%FirstName%** variable. The column header will change to **"%FirstName%"**.

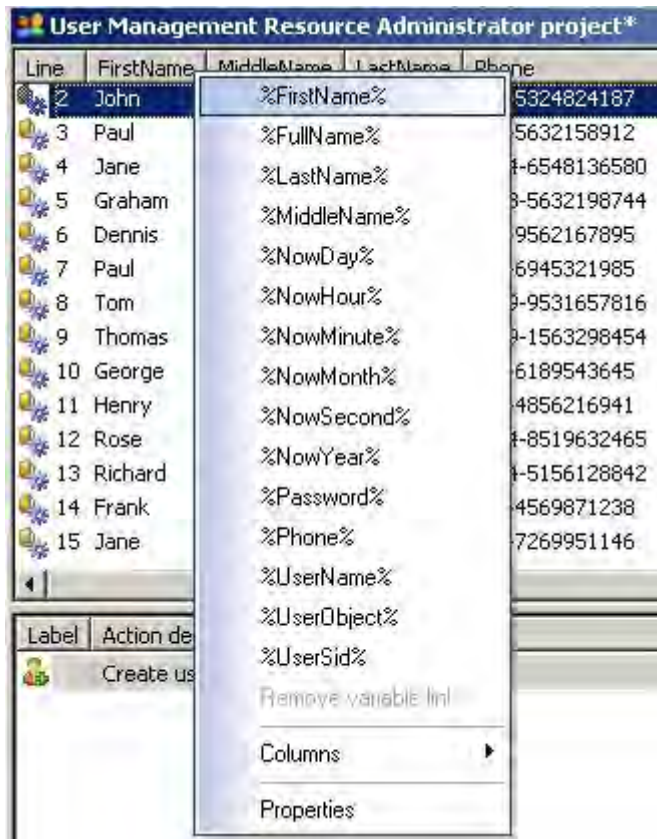


Figure 24: Linking input data columns to variables

Figure 9- Linking input data columns to variables

- Follow the same procedure for the next three columns. Assign the variable **%MiddleName%** to the **"MiddleName"** column, assign the variable **%LastName%** to the **"LastName"** and finally assign the **%Phone%** variable to the **"Phone"** column.

The final result for the four columns is shown in the next figure:










Figure 25: Linking input data columns to variables

Figure 10- Linking input data columns to variables

### Step 3 - Testing the project script

In the **UMRA Console**, there are various options to test and debug your MASS project script. In the toolbar the following icons are related to testing:



Icon	Description
 (Test only)  (Execution mode)	<b>This icon has either the state "Test only" or "Execution mode". Clicking the icon will change its state.</b> In <b>Test only</b> mode, the script will be executed, but without updating Active Directory or other network resources
 (Step mode)	The step mode can be used to execute the project script line by line.
 (Run selection)	The project script will be executed for the selected input data.
 (Run)	The project script will be executed for all input data.

## Running the project script in test mode

In UMRA, a detailed log is generated with the results of each and every executed script action, date and time of script execution, properties set, encountered errors, etc.

1. First of all, make sure that you are running UMRA in **Test only** mode (you should see the icon .

If you see the icon  instead, switch to test mode by clicking the icon).

2. Select the first user in the input data window (John F. Addams) and click the **Run selection** icon (



- OR -

Right-click the first user in the input data window and choose the **Run selection** command.

The full log for the first row being processed is displayed below. The comments in **bold** have been added to explain the individual sections of the log messages.

### Log start showing the build version and the date and time when the job was executed:

Starting User Management Resource Administrator session, build 1233 at 16:25:14 03/07/2006

### Message showing that the script is run in test mode:

16:25:14 03/07/2006 \*\*\*\*\* TEST ONLY \*\*\* STARTING JOB SIMULATION \*\*\* TEST ONLY  
\*\*\*\*\*

### Message indicating the row being processed. It starts with "2" since the first row of the input data contained headers only!:

16:25:14 03/07/2006 \*\*\*\*\* Processing entry 2...

### All the variables which are set for the first row containing a user are listed here. The last variables starting with "%Now(..)%" provide more details on when the individual script actions were executed:

16:25:14 03/07/2006 Variable 1: %FirstName%=John  
16:25:14 03/07/2006 Variable 2: %MiddleName%=F.  
16:25:14 03/07/2006 Variable 3: %LastName%=Addams  
16:25:14 03/07/2006 Variable 4: %Phone%=001-5324824187  
16:25:14 03/07/2006 Variable 5: %NowDay%=07  
16:25:14 03/07/2006 Variable 6: %NowMonth%=03  
16:25:14 03/07/2006 Variable 7: %NowYear%=2006  
16:25:14 03/07/2006 Variable 8: %NowHour%=16  
16:25:14 03/07/2006 Variable 9: %NowMinute%=25  
16:25:14 03/07/2006 Variable 10: %NowSecond%=14

### A listing of all the Create User (AD) script action property values for the first row being processed:




16:25:14 03/07/2006 Creating AD account in specified domain: 't4edoc'.  
16:25:15 03/07/2006 Creating AD account in container 'Users'.  
16:25:15 03/07/2006 Creating AD account in Organizational Unit-Container: 'LDAP://CN=Users,DC=t4edoc,DC=local'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only): Using name generation algorithm 'Default', 100 iterations maximum for duplicate names.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Common name of user set to 'John F. Addams'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). SAM account name (username) of user set to 'addamsjf'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). LDAP attribute 'userPrincipalName' of object 'John F. Addams' set to 'addamsjf@%t4edoc'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). LDAP attribute 'displayName' of object 'John F. Addams' set to 'John F. Addams'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). LDAP attribute 'givenName' of object 'John F. Addams' set to 'John'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). LDAP attribute 'sn' of object 'John F. Addams' set to 'Addams'.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Account disabled'=FALSE (101034). Result not changed.  
16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean



parameter 'Password never expires'=FALSE (101032). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Store password using reversible encryption'=FALSE (101033). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Smart card is required for interactive logon'=FALSE (101035). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Account is trusted for delegation'=FALSE (101036). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Account is sensitive and cannot be delegated'=FALSE (101037). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Use DES encryption types for this account'=FALSE (101038). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Boolean parameter 'Don't require Kerberos preauthentication'=FALSE (101039). Result not changed.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Account expiration date not specified for new user 'John F. Addams' object.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). LDAP attribute 'homePhone' of object 'John F. Addams' set to '001-5324824187'.  
 16:25:15 03/07/2006 Creating AD user account in container/OU  
**In Test mode you will see this line in the log window with the text "Not creating user", indicating that the user is not created for real in Active Directory:**  
 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Not creating user 'John F. Addams' (test only).  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Password set for new user object 'John F. Addams'.  
 16:25:15 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local' (test only). Flag 'User Cannot Change Password' not changed from default value (FALSE).  
**Line indicating that the first user row has been fully processed.**  
 16:25:15 03/07/2006 \*\*\*\*\* Ready processing entry 2...  
**Line showing the total numbers of script execution errors. If any errors occur, the log window will provide more details on the type of error that was encountered.**  
 16:25:15 03/07/2006 Total number of script action execution errors: 0.  
**The text "END TEST ONLY \*\*\*ENDING JOB SIMULATION" will appear at the end of the log file, indicating that the job was executed in test mode:**  
 16:25:15 03/07/2006 \*\*\*\*\* END TEST ONLY \*\*\*\*\* ENDING JOB SIMULATION \*\*\*\*\* END TEST ONLY \*\*\*\*\*  
 16:25:15 03/07/2006 \*\*\*\*\* (Select Actions, Test only, to toggle TEST ONLY on and off) \*\*\*\*\*  
 End of session

When no errors are reported in the log messages window, you are ready to run the project script in execution mode to update Active Directory.

#### Step 4 - Executing the project script

1. Switch to execution mode (you should see the icon ). If you are still running the **UMRA Console** in Test mode (icon displayed as ) , just click the icon to change its state.
2. Click the **Run** icon (  ) to execute the script for all input data. The user accounts will now be created for real.

All the script execution details can again be traced in the log messages window. Note that the text "text **"END TEST ONLY \*\*\*ENDING JOB SIMULATION"** has disappeared in execution mode. The complete log for the first row being processed now looks as follows. Once again, the comments in **bold** have been added to explain the individual sections of the log messages.

#### **Log start showing the build version and the date and time when the script was executed:**

Starting User Management Resource Administrator session, build 1233 at  
 16:37:02 03/07/2006

#### **Message indicating the row being processed. It starts with "2" since the first row of the input data contained headers only!:**

16:37:02 03/07/2006 \*\*\*\*\* Processing entry 2...

**All the variables which are set for the first row containing a user are listed here. The last variables starting with "%Now(.)" provide more details on when the individual script actions were executed:**

```
16:37:02 03/07/2006 Variable 1: %FirstName%=John
16:37:02 03/07/2006 Variable 2: %MiddleName%=F.
16:37:02 03/07/2006 Variable 3: %LastName%=Addams
16:37:02 03/07/2006 Variable 4: %Phone%=001-5324824187
16:37:02 03/07/2006 Variable 5: %NowDay%=07
16:37:02 03/07/2006 Variable 6: %NowMonth%=03
16:37:02 03/07/2006 Variable 7: %NowYear%=2006
16:37:02 03/07/2006 Variable 8: %NowHour%=16
16:37:02 03/07/2006 Variable 9: %NowMinute%=37
16:37:02 03/07/2006 Variable 10: %NowSecond%=02
```

**A listing of all the Create User (AD) script action property values for the first row being processed:**

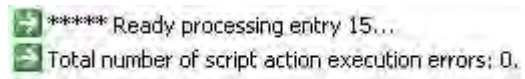
```
16:37:02 03/07/2006 Creating AD account in specified domain: 't4edoc'.
16:37:02 03/07/2006 Creating AD account in container 'Users'.
16:37:02 03/07/2006 Creating AD account in Organizational Unit-Container: 'LDAP://CN=Users,DC=t4edoc,DC=local'.
16:37:02 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local': Using name generation
algorithm 'Default', 100 iterations maximum for duplicate names.
16:37:02 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Common name of user
set to 'John F. Addams'.
16:37:02 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. SAM account name
(username) of user set to 'addamsjf'.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. LDAP attribute
'userPrincipalName' of object 'John F. Addams' set to 'addamsjf@%t4edoc'.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. LDAP attribute
'displayName' of object 'John F. Addams' set to 'John F. Addams'.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. LDAP attribute
'givenName' of object 'John F. Addams' set to 'John'.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. LDAP attribute 'sn' of
object 'John F. Addams' set to 'Addams'.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter
'Account disabled'=FALSE (101034). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter
'Password never expires'=FALSE (101032). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter 'Store
password using reversible encryption'=FALSE (101033). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter
'Smart card is required for interactive logon'=FALSE (101035). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter
'Account is trusted for delegation'=FALSE (101036). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter
'Account is sensitive and cannot be delegated'=FALSE (101037). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter 'Use
DES encryption types for this account'=FALSE (101038). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Boolean parameter 'Don't
require Kerberos preauthentication'=FALSE (101039). Result not changed.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Account expiration date
not specified for new user 'John F. Addams' object.
16:37:03 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. LDAP attribute
'homePhone' of object 'John F. Addams' set to '001-5324824187'.
```

**Line showing that the user account was created for real in Active Directory:**

```
16:37:04 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. User 'John F. Addams'
successfully created.
16:37:04 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Password set for new
user object 'John F. Addams'.
16:37:04 03/07/2006 Creating AD user account in container/OU 'LDAP://CN=Users,DC=t4edoc,DC=local'. Flag 'User Cannot Change
Password' not changed from default value (FALSE).
16:37:04 03/07/2006 ***** Ready processing entry 2...
```

As mentioned at the beginning of this section, only a small section of the log has been displayed here. In the full log as shown in the **Log Messages** window, you can trace the processing results of the remaining

rows of the input data. At the very end of the log a message is displayed indicating how many rows in total have been processed (in this case 15):

A screenshot of a log window with a light gray background. It contains two lines of text, each preceded by a small green icon of a document with a right-pointing arrow. The first line reads "\*\*\*\*\* Ready processing entry 15..." and the second line reads "Total number of script action execution errors: 0.".

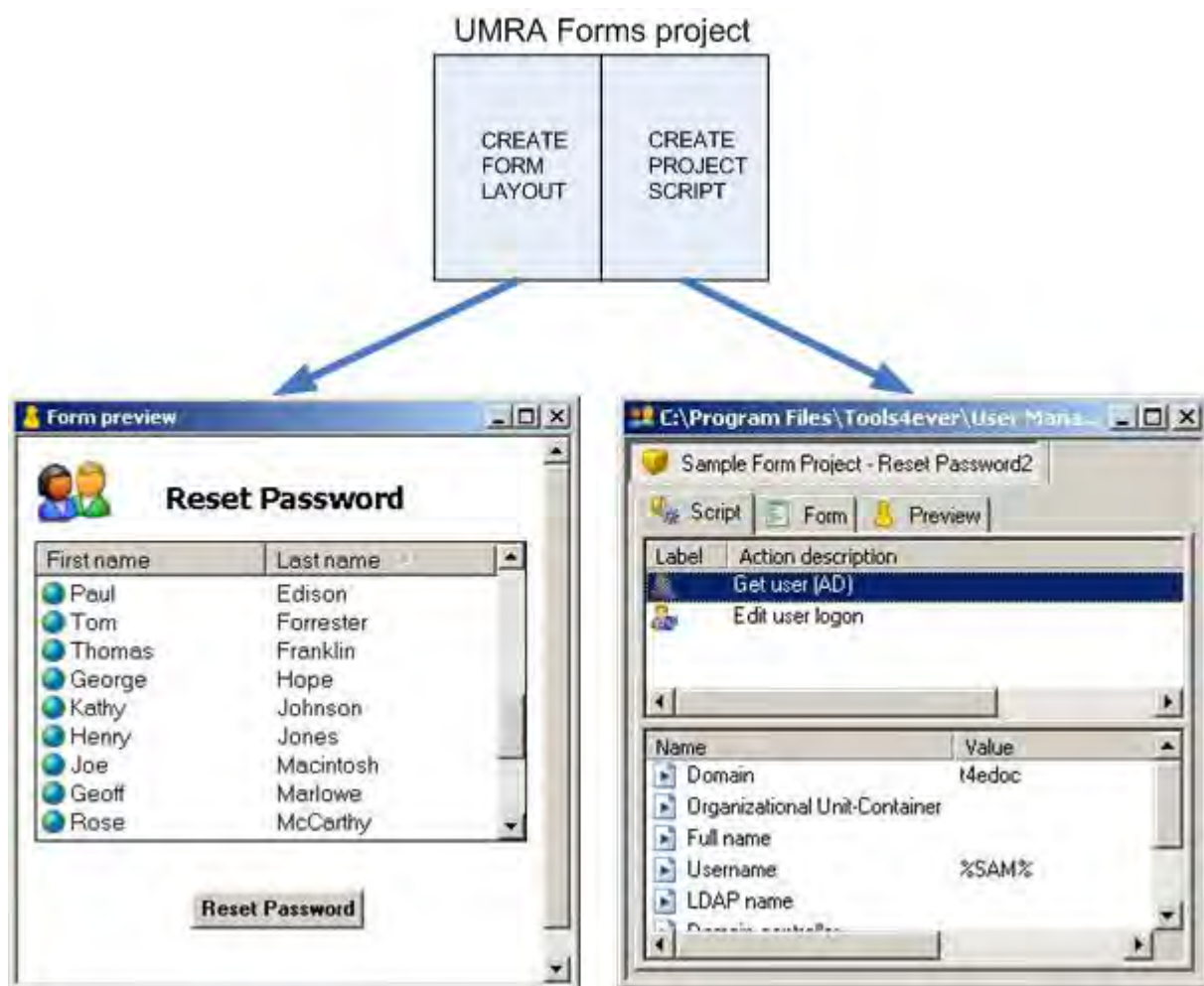
```
***** Ready processing entry 15...  
Total number of script action execution errors: 0.
```

### 3.2.8. Creating a Forms example project - Reset password

To demonstrate the various steps involved in creating a delegation solution, we will create a sample project to delegate the task of resetting a user's password to a non-admin.

To create the project, the same steps will be followed as described in chapter *Delegating user account management tasks* on page 4:

1. Creating the form layout
2. Building the form script
3. Specifying security



In the first step, the interface for the end user will be designed. In this particular form, a user can be selected from a list. When the delegated user hits the **Reset Password** button, a project script is executed which resets the password for the selected user. This script is explained in detail in step 2. In the last step we will show you how the security for the project is defined. In other words: how to control access to the project form.

Please note that this project only serves as an example. It can be extended with many more script actions and the form can be fully customized to your own specific needs. For a full list of script actions, see *Appendix A - Script actions* on page 64.

**Introduction**

Using UMRA, administrators can create a project to delegate specific user account management tasks (e.g. resetting a password) to non-admins. The non-admin is only presented with a simple interface (a form) to perform this specific task and is not given an admin password.

For any delegation project, a design phase and an operational phase can be distinguished.

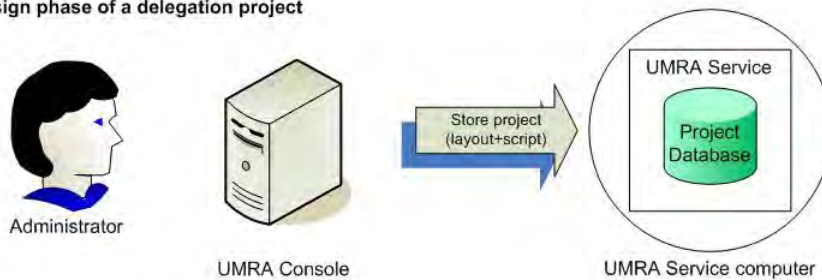
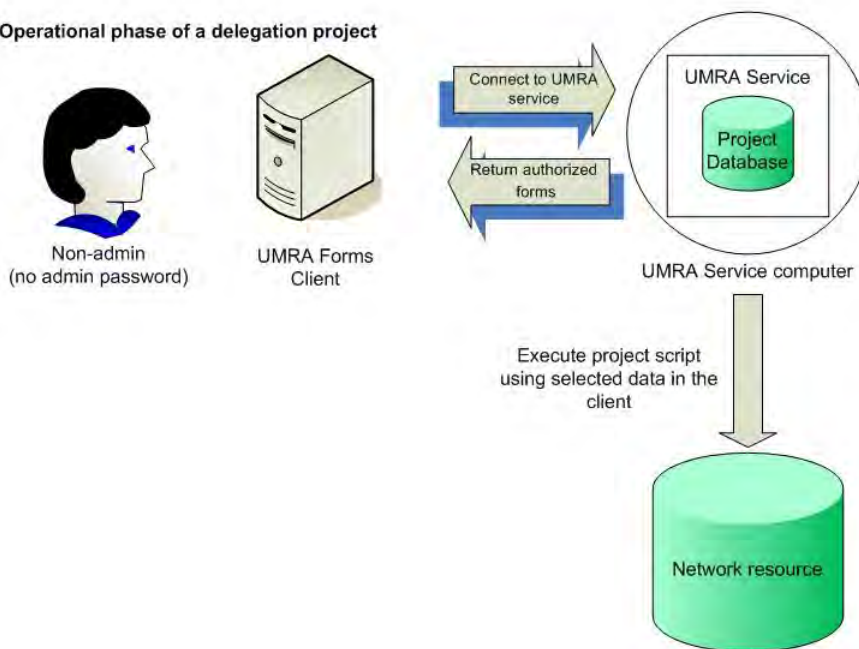
**Design phase of a delegation project****Operational phase of a delegation project**

Figure 26: The design and operational phase of a delegation project

In the design phase, the administrator creates an UMRA project in the **UMRA Console** to delegate a specific user management task to a non-admin. This delegation project consists of a project form to present to the delegated user and a project script specifying the actions to be executed.

The complete project is stored in a project database which is maintained by the **UMRA service**. This principle is shown in Figure 18.

In the operational phase, the delegated user connects to the **UMRA service** using the delegation client (**UMRA Forms**). The **UMRA service** will check the access rights of the user and returns a list of project

forms for which the delegated user has been given privileges. These will be displayed in the **UMRA Forms** client.

It is important to understand that the form project as presented in the **UMRA Forms** client does not contain any scripting. The project script, developed by the administrator, has been separated from the form and is part of the project maintained by the **UMRA service**.

As soon as a delegated user hits an action button (e.g. Reset Password) in the project form on the client side, a request is sent to the **UMRA service** to execute the project script for this project using the data specified in the **UMRA Forms** client (e.g. resetting the password for the selected user in the form). The network resource will then be updated accordingly.



Figure 27: Example of an UMRA form to delegate a user account management task

**Step 1 - Designing the form layout**

The form you need to create will be the main interface for the end user to execute a specific user management task. You can safely assume that this end user does not have any knowledge of Active Directory. The form layout should therefore be created in such a way that the purpose of the form is crystal clear. In the **UMRA Console**, various form field categories are available to customize the interface for your own specific needs:

- **Descriptive form fields** - Use the static text field to explain the purpose of the form and the individual form elements to the end user. Examples: form title, input box description ("Please enter the domain name"), etc.
- **Interactive form fields** - Use input text boxes, checkboxes, radio buttons and tables to allow for interaction with the end user. Examples: Allowing the end user to enter the domain name in an input box, presenting a list of users in Active Directory which can be selected, etc.
- **Cosmetic form fields** - Use the picture (e.g. to add a company logo) and vertical space form fields to make your form more appealing.



To include these field elements in a form, all you need to do is drag the required form element from the **Field elements** window into the project's **Form** window and configure its display properties.

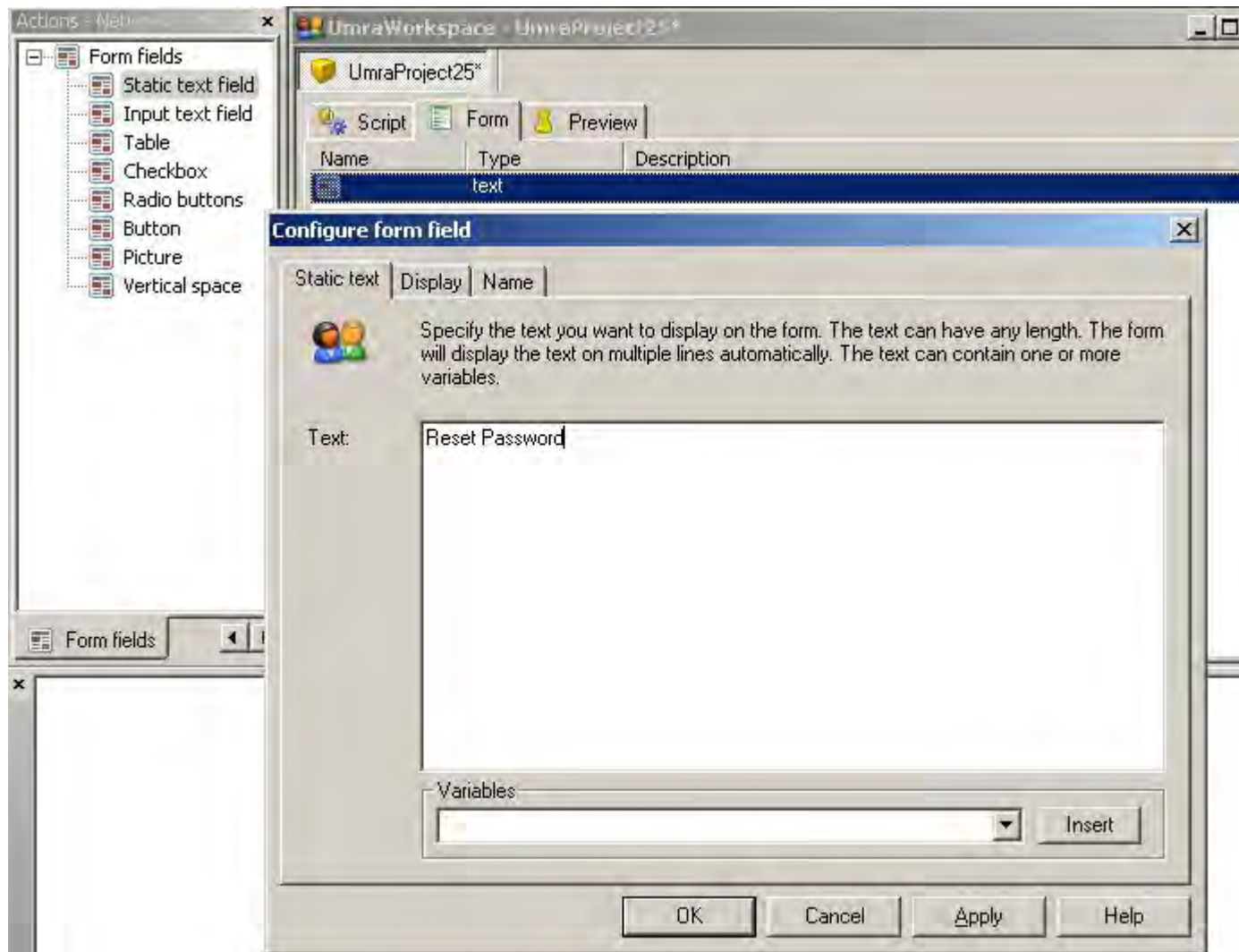


Figure 28: Adding a title with the Static text field form element

Figure 1 - Adding a title with the Static text field form element

The various field elements for the **Reset Password** form are listed in figure 22.

Figure 29: Form field elements for the Reset Password project

Figure 2 - Form field elements for the Reset Password project

- 1 - Picture
- 2 - Static text field for form title
- 3 - Generic table containing data from an LDAP query
- 4 - Static text field
- 5 - Input text field
- 6 - Static text field
- 7 - Input text field
- 8 - (Execute) button

1. This and the next section(s) explain how you can build a form project, by describing in steps how to create an example "Reset password" project. A finished reset password project that should be comparable to the results you will get from following the instructions is available. Select the menu UMRA service, Manage Server Projects. This gives a list of the currently available projects on the service. If the project is not listed here, you can import it by selecting the import button and browsing to the program directory of the console and look here in either the "Projects" or "sample projects" directory.

**Starting a new form project**

1. Start the **UMRA Console**. If you have previously installed the UMRA Service, UMRA will automatically establish the connection. If this is not the case, then please install the **UMRA Service** first (see also *Appendix B - Installing the UMRA Service* on page 69).
2. Select **File**→**New** and choose the option **Form project** to start a new form project.

**Adding the form field elements****a** Picture (1)

1. Click the **Form fields** tab in the **Actions-Network-Form fields** window and drag the **Picture** form field to the project's **Form** window. The **Configure form field** window will appear in which you can configure the display properties for this form field.
2. In the **File** field under **Image file name**, specify the name of the picture you wish to use for the form.

**b** Vertical space

1. Click the **Form fields** tab in the **Actions-Network-Form fields** window and drag the **Vertical space** form field to the project's **Form** window. The **Configure form field** window will appear in which you can configure the display properties for this form field. In the **Amount of pixels moved down** field, enter **"15"**.

**c** Static text field (2)

1. Click the **Form fields** tab in the **Actions-Network-Form fields** window and drag the **Static text field** form field to the project's **Form** window. The **Configure form field** window will appear.
2. Enter the title “**Reset Password**” in the **Text** section. This is the title of your form.

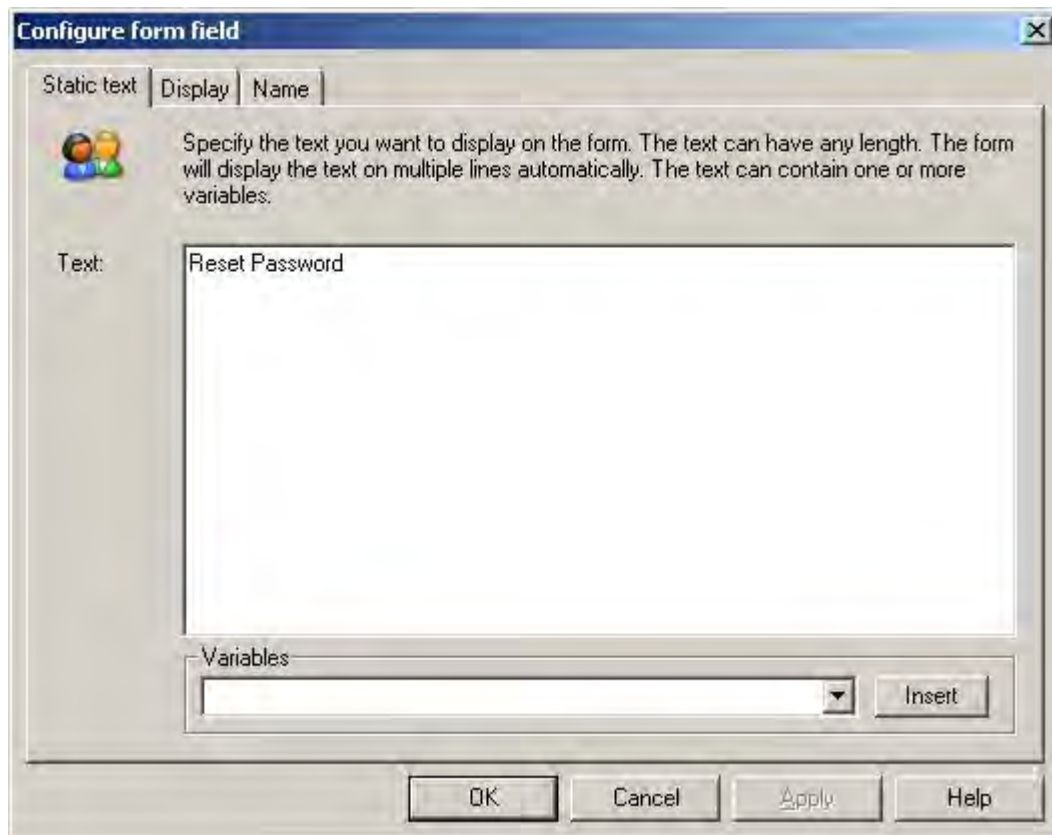
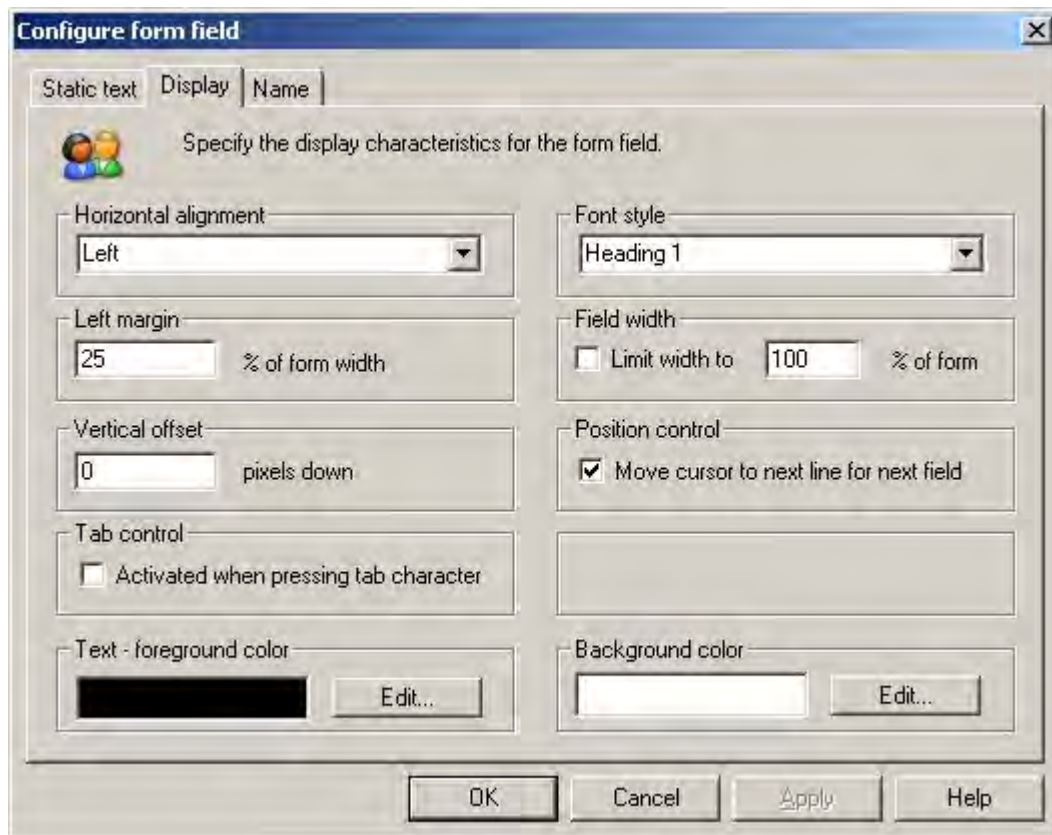


Figure 30: Entering text for the form title (static text form field element)

Figure 3 - Entering text for the form title (static text form field element)

- Click the **Display** tab in the **Configure form field** window and set the display configuration of the title text as shown in figure 24.



4.

Figure 31: Specifying the display configuration for a static text field

Figure 4 - Specifying the display configuration for a static text field

#### d Vertical space

Create a vertical space of 15 pixels.

#### e Table (3)

For this project, a table will be inserted to display a list of users from Active Directory. Since it requires some additional steps to configure a table, it is discussed separately.

- Follow the instructions as described in the following section, "Tables - Specifying an LDAP table to create a list of users". For more information about tables in UMRA, see the UMRA Tables User Guide.

#### f Vertical space

Create a vertical space of 10 pixels

#### g Static text field (4)

Create a static text field with title "Password" and a left margin of 0%.

**h** Input text field (5)

Create an Input text field and specified as variable name %PassWord%.

When the at run time the form is submitted to the service, the value entered in this input field will be stored in the mentioned variable. The variables will be used in the script as input for the actual actions.

**Configure form field**

Input | Display | Name

Specify the settings for this input text field.

Initial text

Text:

Variables:

Appearance

☐ Text field supports multiple lines with  visible lines

☒ Margin between field border and text of  pixels

☒ Password style, all characters shown as an asterisk (\*)

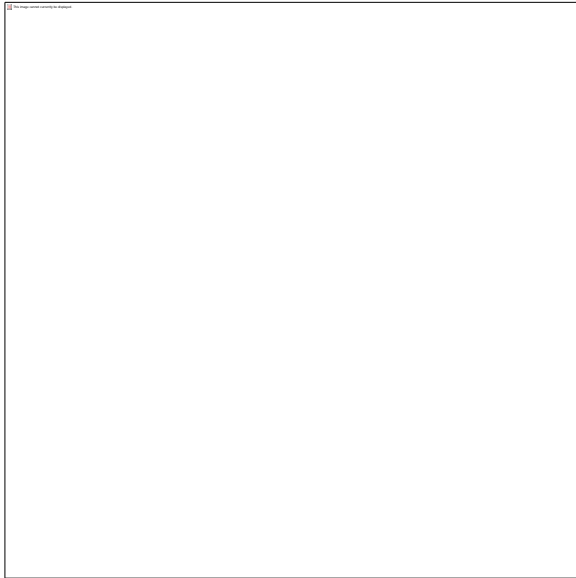
☒ Draw border of input field

☐ Accepts carriage return (<Enter>) characters

Variable

On submit, store contents in variable:

Figure 5 - Configure Input form field



**i** Vertical space

Create a vertical space of 5 pixels

**j** Static text field (6)

Create a static text field with title "Confirm password" and a left margin of 0%.

**k** Input text field (7)

Create an input text field .

**l** Vertical space

Create a vertical space of 5 pixels

**m** Button (8)

This is the execute button for the form. When it is pressed, the form is submitted to the **UMRA service** and an action will be executed (e.g. "execute the project script"). To configure a button, you will therefore need to specify both its display options and the actions to be executed upon clicking.

1. Click the **Form fields** tab in the **Actions-Network-Form fields** window and drag the **Button** form field to the project's **Form** window. The **Configure form field** window will appear.
2. In the **Button text** field under **Appearance**, enter the button display text "**Reset Password**" figure.



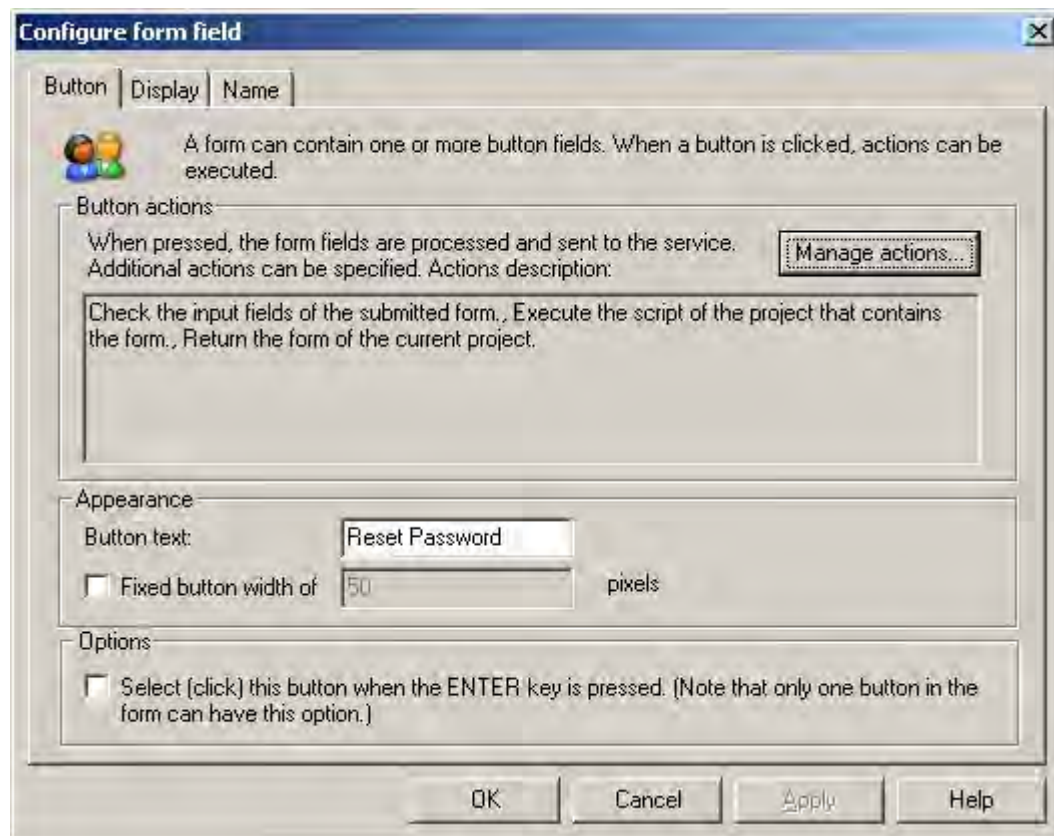


Figure 32: Defining the button text

Figure 5 - Defining the button text

3. Click the **Display** tab of the **Configure form field** window and set the button display properties as shown in figure 26. Note that the **Horizontal alignment** option determines the alignment of the *text* within the form field and not the form field itself.



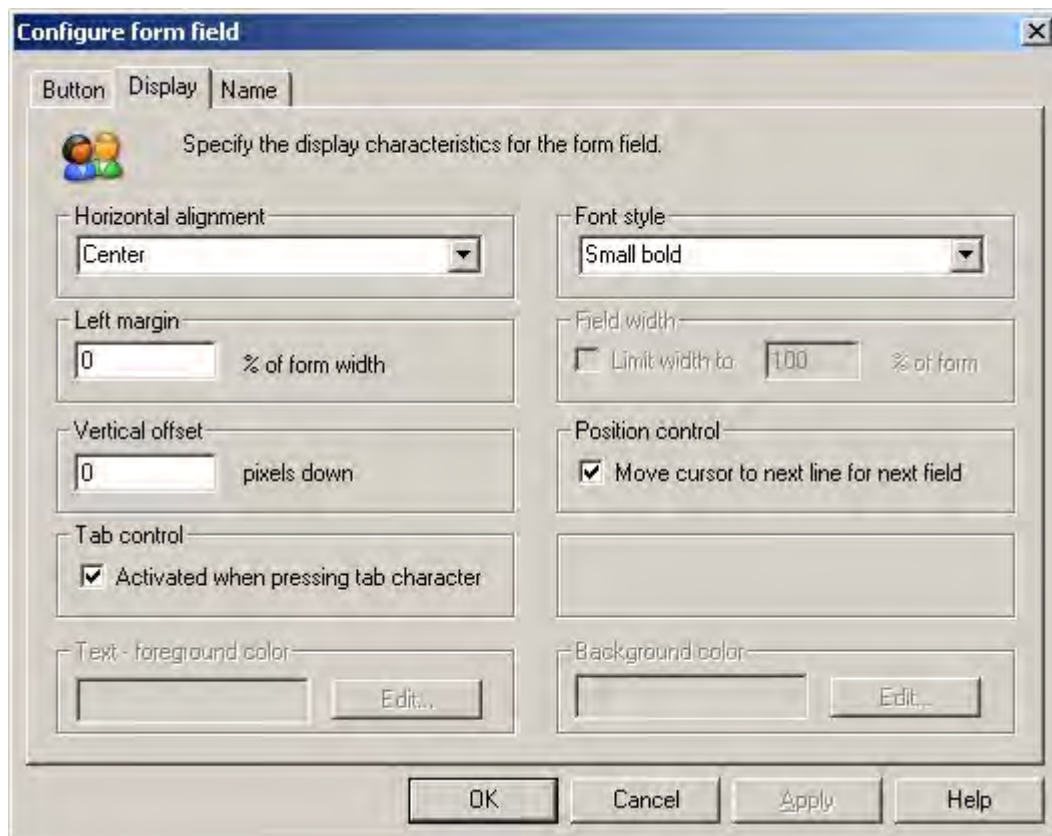


Figure 33: Configuring the display of the button and the button text.

Figure 6 - Configuring the display of the button and the button text.

The above mentioned steps for specifying a button only cover the display characteristics of the button itself. Next, we need to specify which actions should be executed when the user hits the **Reset Password** button.

4. Click the **Manage actions** button and select the following actions:
5. Execute the script of the project that contains the form - this specifies that the project script of the current form will be executed as soon as the delegated user hits the **Reset Password** button
6. Return the form of the current project - this specifies that the current form needs to be displayed again once the project script execution has been completed.

### Tables - Specifying an LDAP table to create a list of users

In a form, the administrator can also add form tables. Form tables are used to display user resource data from Active Directory and other information systems. In figure 27 for instance, an LDAP table is shown displaying users in Active Directory.



Figure 34: The marked area shows an LDAP table displaying users in Active Directory.

*Figure 7 - The marked area shows an LDAP table displaying users in Active Directory.*

1. When the delegated user selects a table entry and hits the **Reset Password** button, the project script needs to be executed for the selected user. Figure 28 illustrates how this principle works. The LDAP table shows columns for the user's first name, last name and sAMAccountname. The **SAM Account Name** column has been linked to the **%SAM%** variable. When the user selects a row (e.g. "Thomas Franklin") and hits the **Reset Password** button, the project script is executed using the selected data. The project script includes the **Get User** script action to find the user account for the selected user. In this example (there are multiple ways to obtain a user account), this script action uses the **sAMAccountName** of the selected user to obtain the LDAP name of the user account (e.g. LDAP://CN=Thomas Franklin,CN=Users,DC=t4edoc,DC=local).
2. The sAMAccountName is specified in the script action's **Username** property value with the variable **%SAM%**. When the script is executed, this variable is replaced with the sAMAccountName of the selected user (franklint).
3. Note that in the final form layout, you would probably want to hide the **SAM Account Name** column, since it provides no useful information for the delegated user.

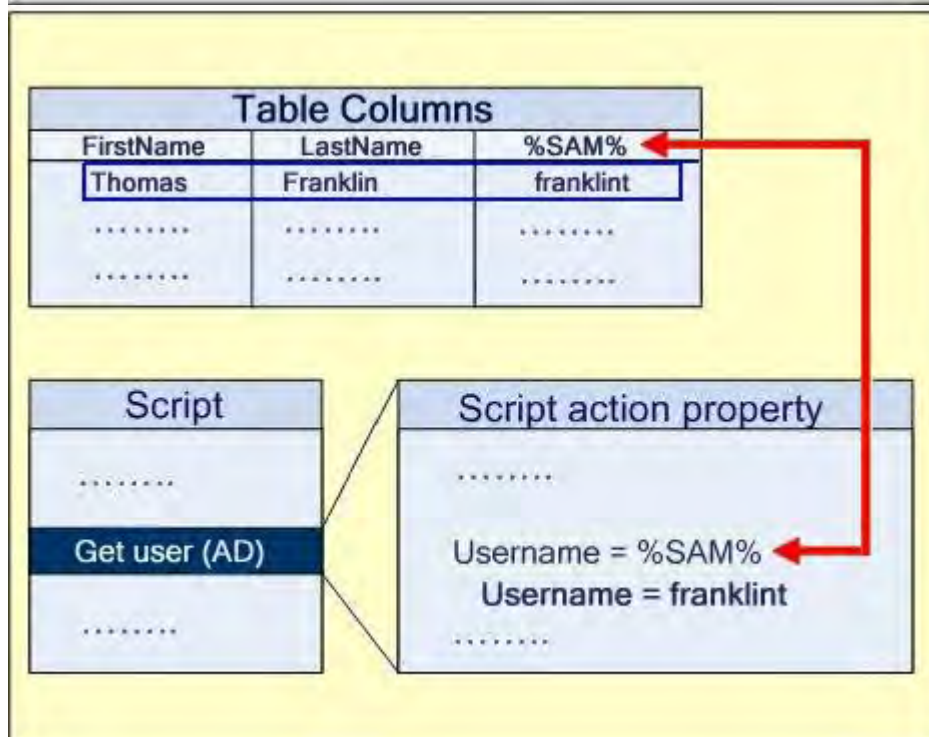
**Form preview**

## Reset Password

Select the user for whom the password must be reset and click the Reset Password button:

First name	Last name	SAM Account Name
Thomas	Franklin	franklint
John	Addams	addamsjf
George	Hope	hopeg
Jane	Austin	austinj
Henry	Jones	joneshp
Dennis	Cutler	cutlerd
Rose	McCarthy	mccarthy
Tom	Forrester	forresterte
Richard	Miles	milesr
Graham	Bell	bellg

**Reset Password**



To create an LDAP table in a form project, the following steps should be followed:

1. **Specifying the table type** - select the required table type

2. **Specifying the LDAP binding** - the scope of the LDAP search;
3. **Specifying an LDAP filter** - the objects you wish to filter on;
4. **Specifying the attributes** - the attributes you wish to retrieve for these objects.
5. **Testing the LDAP query**

**Note:** For a full introduction to using tables in UMRA, see the *UMRA Tables User Guide*.

#### Specifying the table type

1. Doubleclick the table form field and select the **Generic table** option.
2. Click the **Configure** button

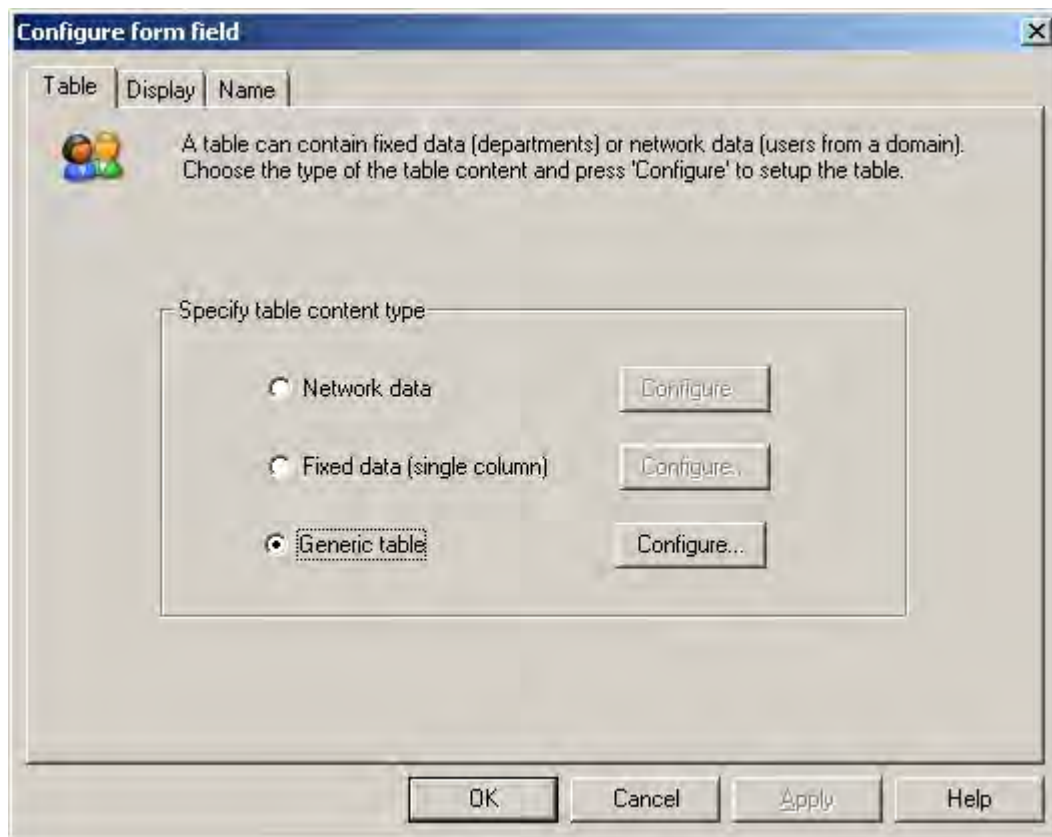


Figure 35: Specifying a table type

Figure 9 - Specifying a table type

3. Click the **Configure** button again. A dialog box is shown where you can select the table type.

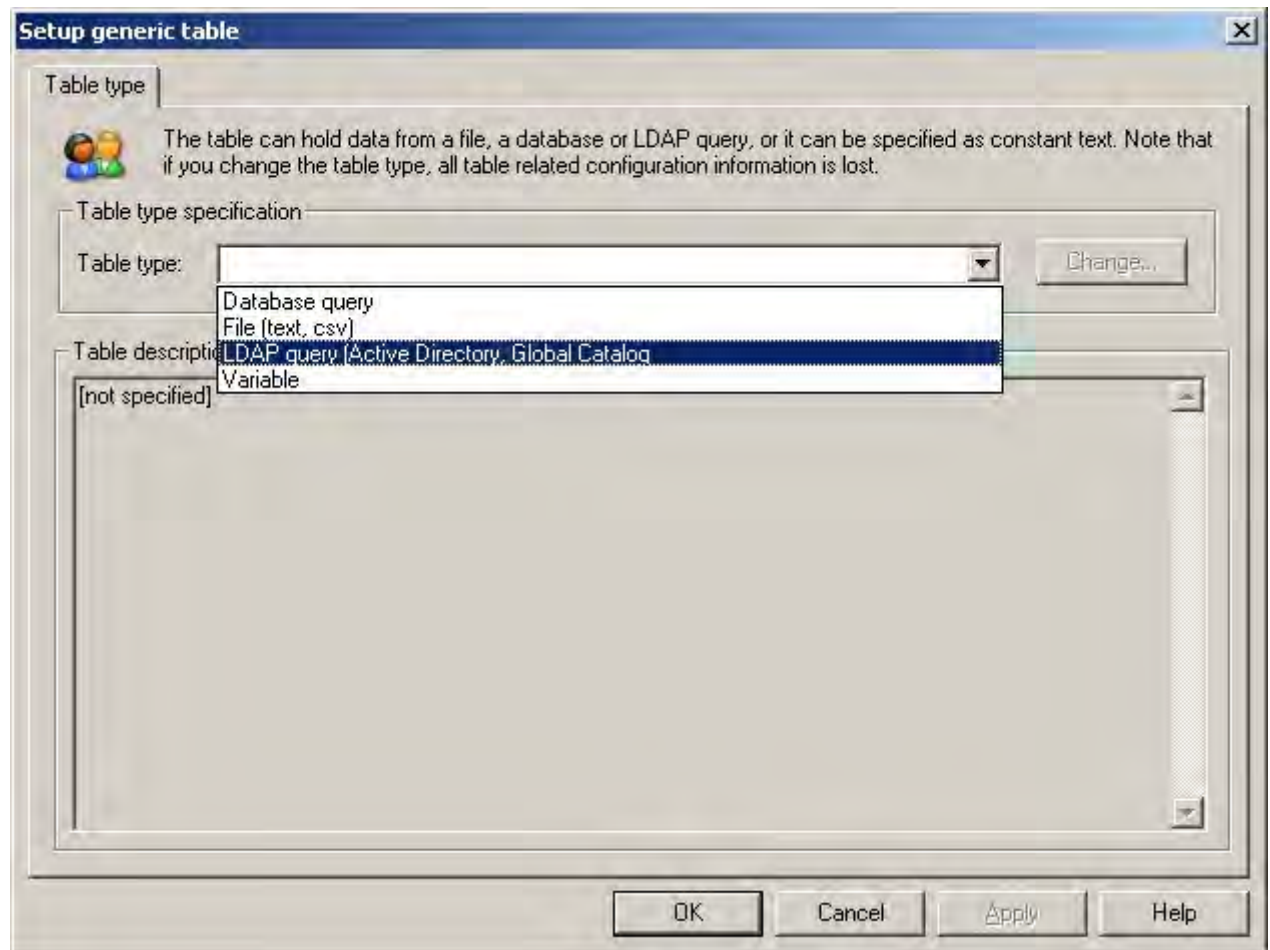


Figure 36: Selecting the LDAP generic table

Figure 10 - Selecting the LDAP generic table



4. In the **Table type** list box, select the table type **LDAP query**. Several tabs will now appear for setting up the LDAP table.

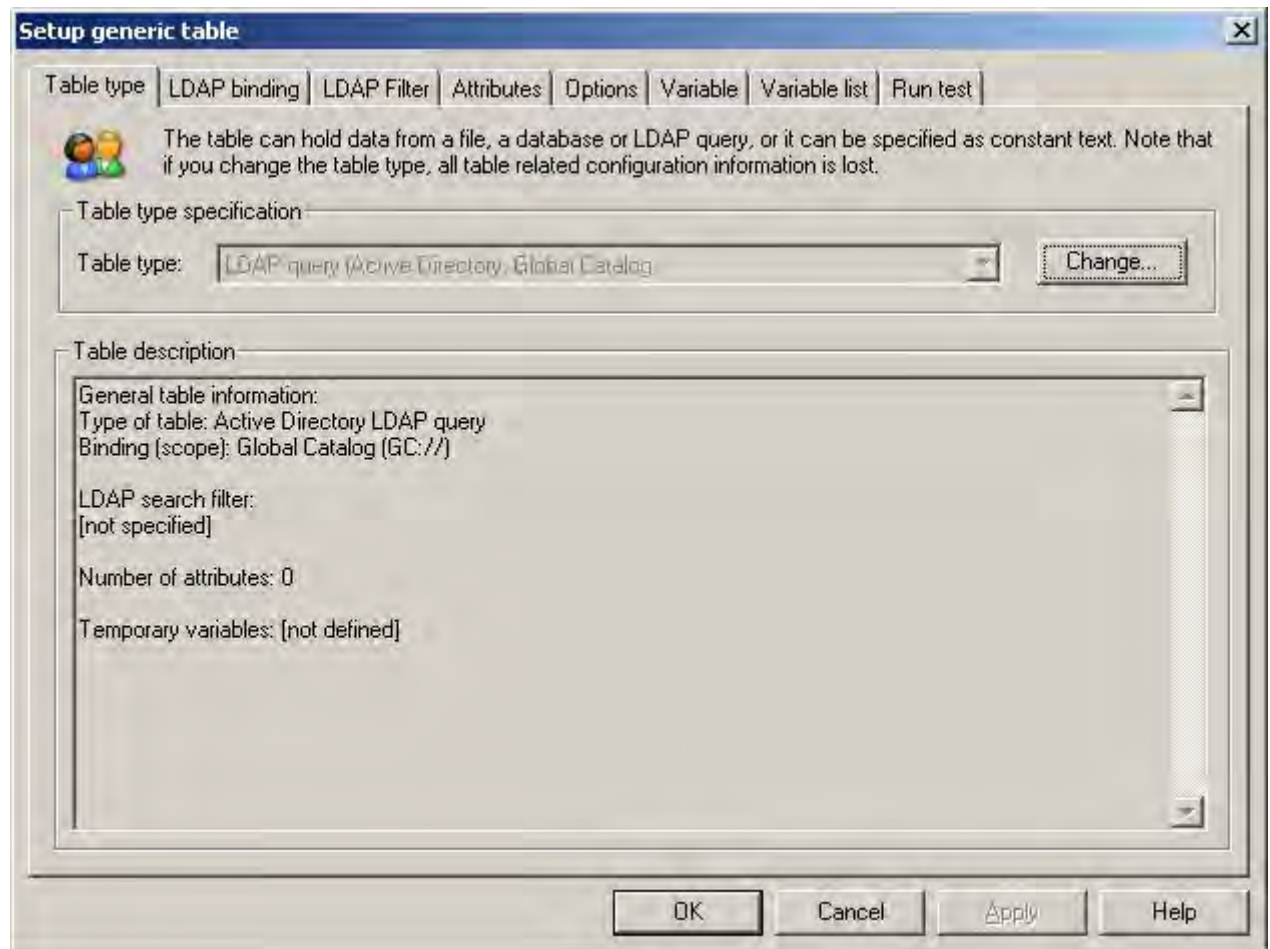


Figure 37: Configuration window for an LDAP table

Figure 11 - Configuration window for an LDAP table

### Specifying the LDAP binding

Starting with Windows 2000, the LDAP provider is used to access Active Directory. This binding method requires a binding string. You can either specify this string manually or select an automatic binding method.

1. Click the **LDAP binding** tab and accept the default binding method (**Global Catalog**).

2. Click **OK**.

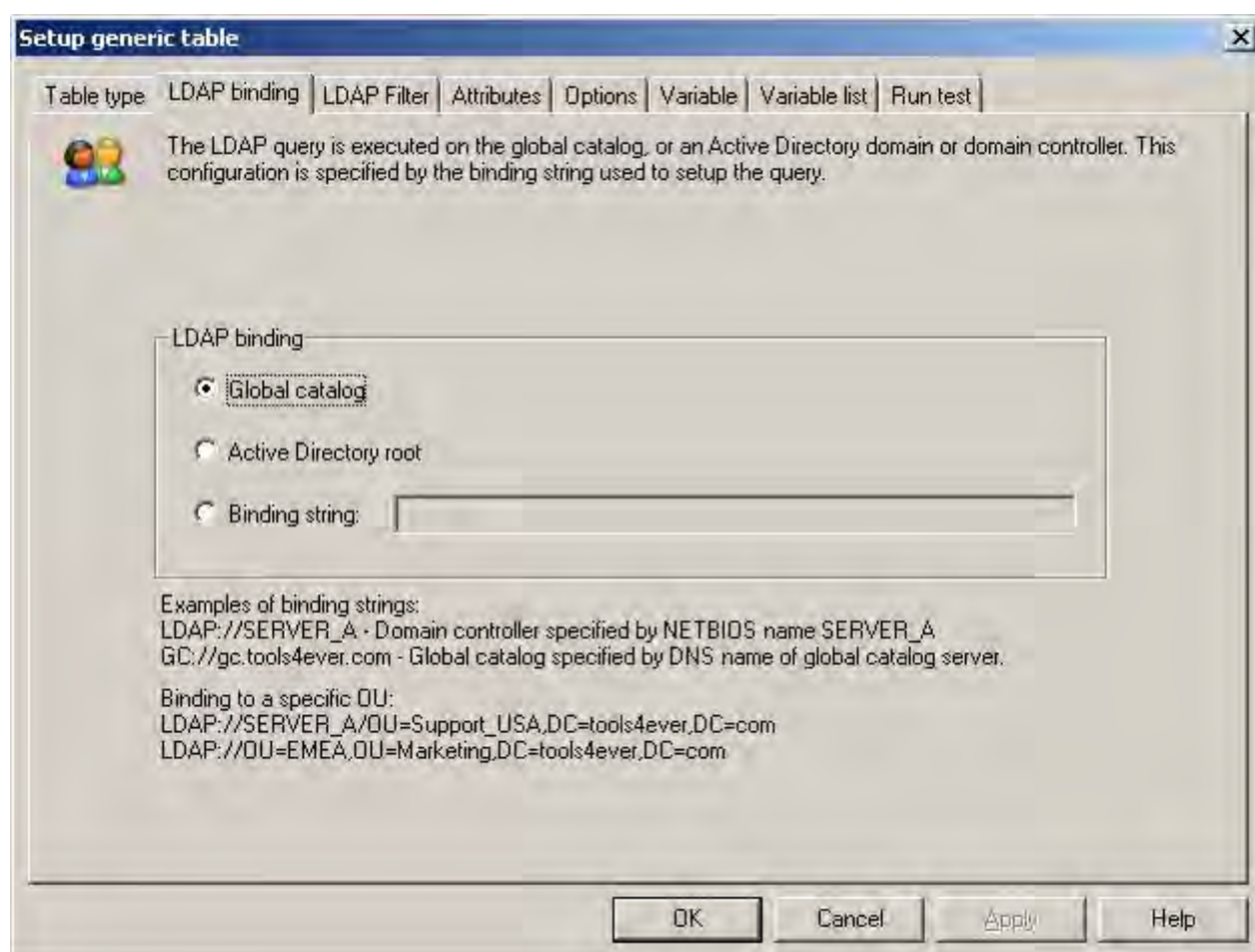


Figure 38: Specifying the binding method

Figure 12 - Specifying the binding method

### Specifying an LDAP filter

An LDAP search filter can be defined as a clause specifying the conditions that must be met by Active Directory objects. Only those objects meeting the requirements will be returned. For our sample project, we want to filter on all users (objectClass=user). Computer accounts and built-in accounts however, which also belong to the user objects class, should not appear in the list. We will therefore include a condition stating that the user object should have a surname (sn=\*).

1. Click the **LDAP Filter** tab.
2. Enter the following clause in the **LDAP search filter** window:

(&(objectClass=user)(sn=\*))

This LDAP query will retrieve all objects of the user class with a surname.

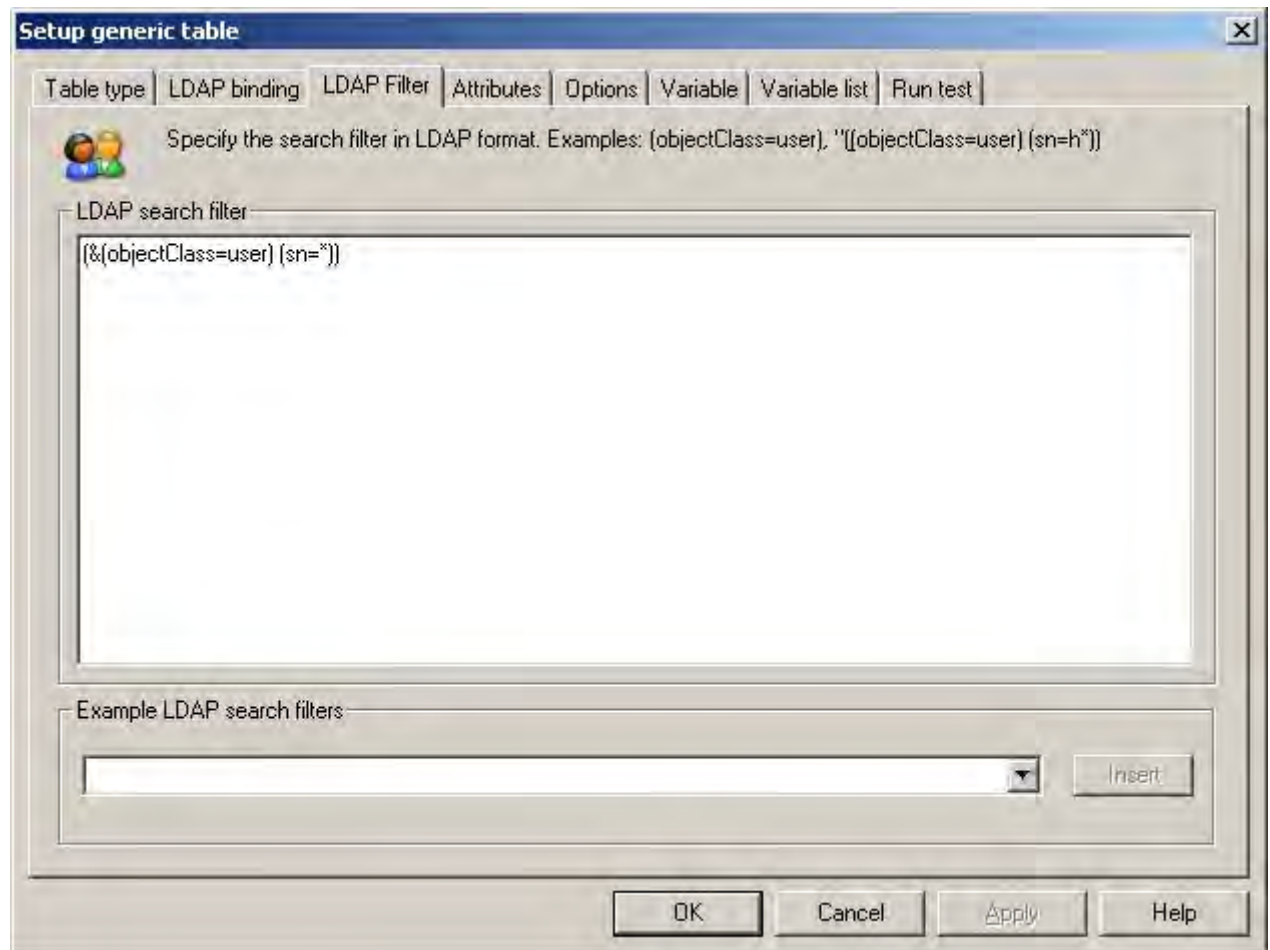


Figure 39: Specifying an LDAP filter

### Specifying attributes

Each object in Active Directory has a set of attributes, defined by and depending on its type and class. The LDAP filter defined in the previous step will filter on user objects. In the **Attributes** tab you can define the LDAP display name for the attributes you wish to return for these filtered objects.

1. Click the **Attributes** tab.



- From the **Default attribute settings** list, select the option **Users - names**. A list of attributes will appear in the **Attributes** list.

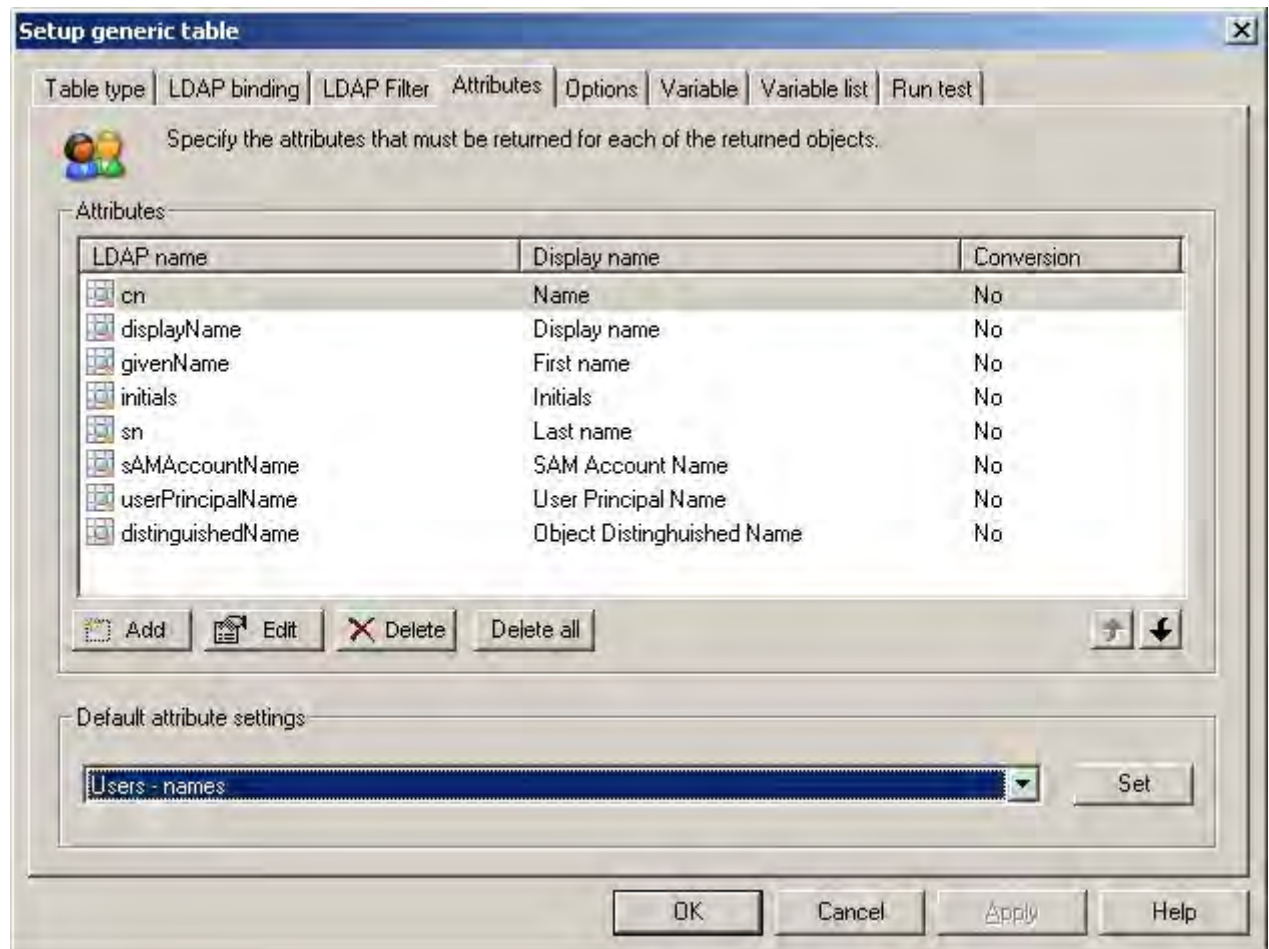


Figure 40: Specifying the LDAP attributes for the LDAP query

- For this example project, we only need some of the listed attributes. Only the following attributes should be kept:

givenName, sn and sAMAccountname.

The first two attributes represent the first and last name of the user. The **sAMAccountName** is the user's logon name. In the second part of this example, it will be used to obtain the user account of the selected user.

4. Delete the other attributes by selecting them and clicking the **Delete** button.

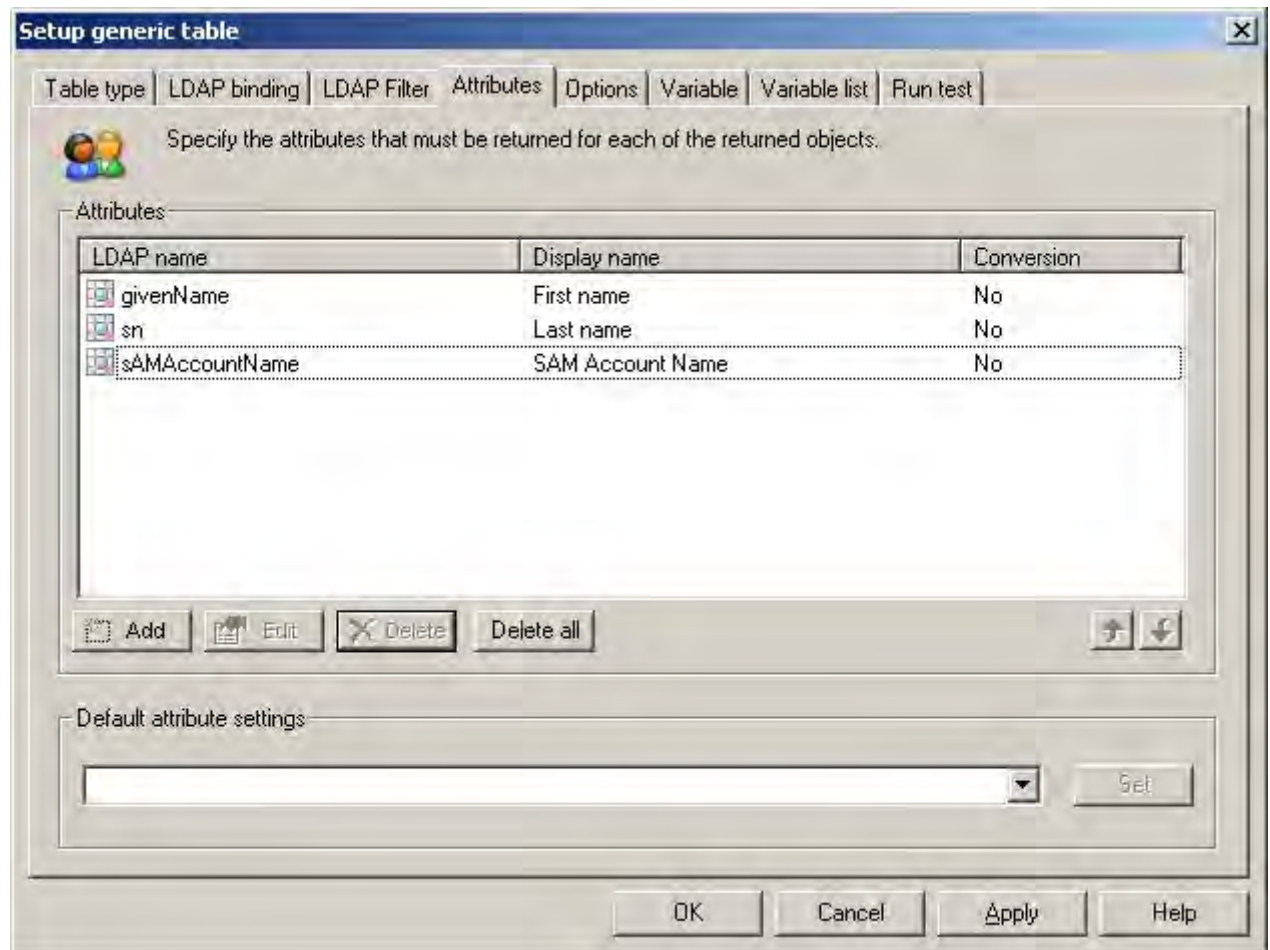


Figure 41: Final list of attributes for the LDAP query

### Testing the LDAP query

1. Click the **Run test** tab and click the **Test** button. The resulting data should be listed in the **Table data** window. If not, revisit the previous steps and make sure you have configured the LDAP table correctly.

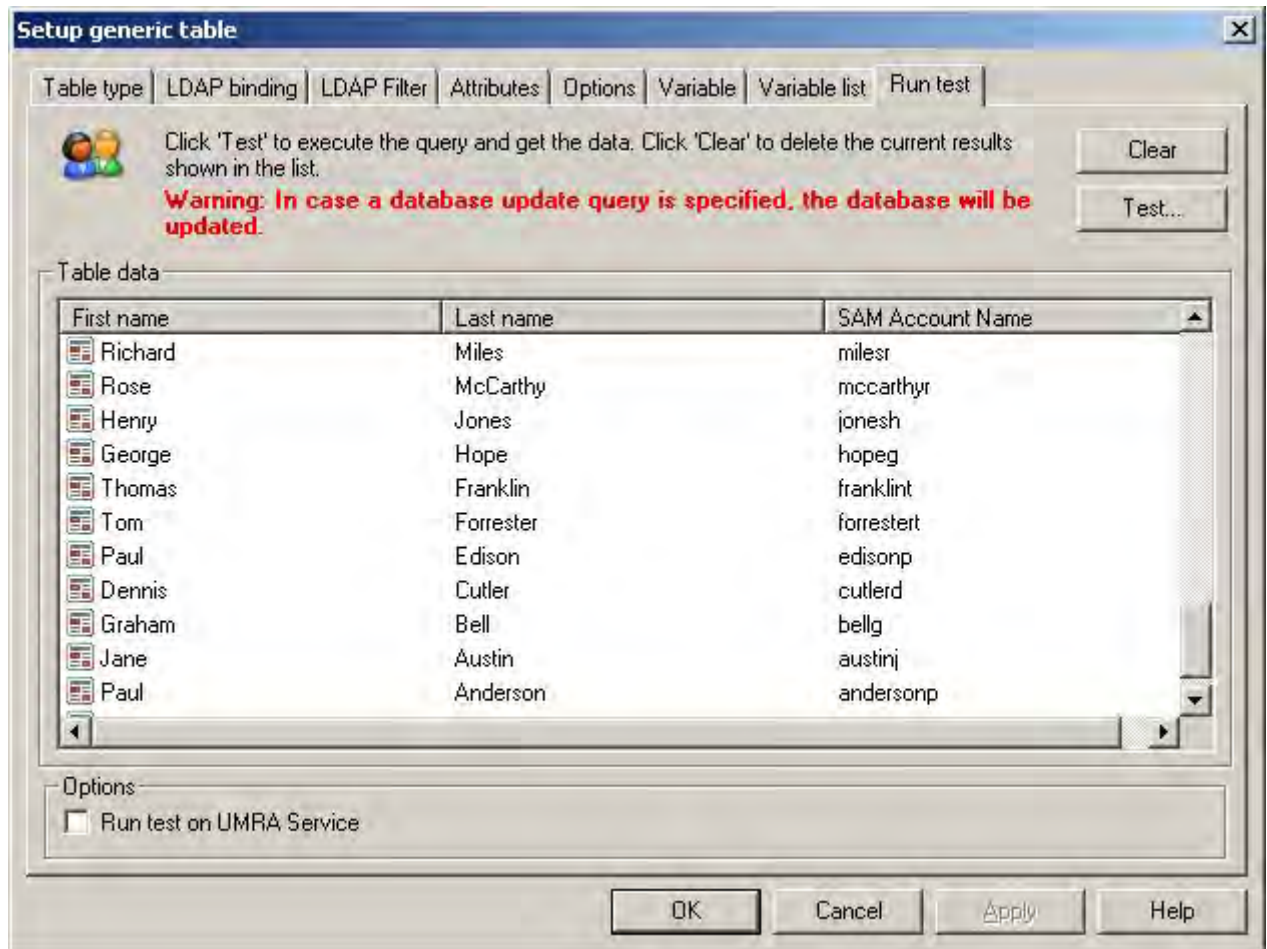


Figure 42: Testing the LDAP query

2. Click **OK** when the test was successful.

### Linking a table column to a variable

The defined LDAP table will retrieve the users' first name, last name and sAMAccountName. The next step is to make it possible to feed the selected user to the project script which we will define in part 2. To do this, we need to link the LDAP table column containing the user's logon name (sAMAccountName) to a variable. This variable can then be used in the project script to obtain the user account of the selected user.

1. Click the **Columns** tab.

2. Select the columns to be displayed in the table: **First name**, **Last name** and **SAM Account Name**. Click the right-arrow to move these columns to the **Current column configuration** window.
3. Select **sAMAccountName** under **Current column configuration**.
4. Enter the variable name **%SAM%** in the **Variable** list box to assign this column to the variable **%SAM%**.
5. Since the **sAMAccountName** is not relevant for the delegated user, this column will be hidden. Set the **Column width** to "0".
6. In the same way, link the **First name** column to the **%FirstName%** variable and the **Last name** column to the **%LastName%** variable, **but do not change the column width**. The final result is shown in the figure below.

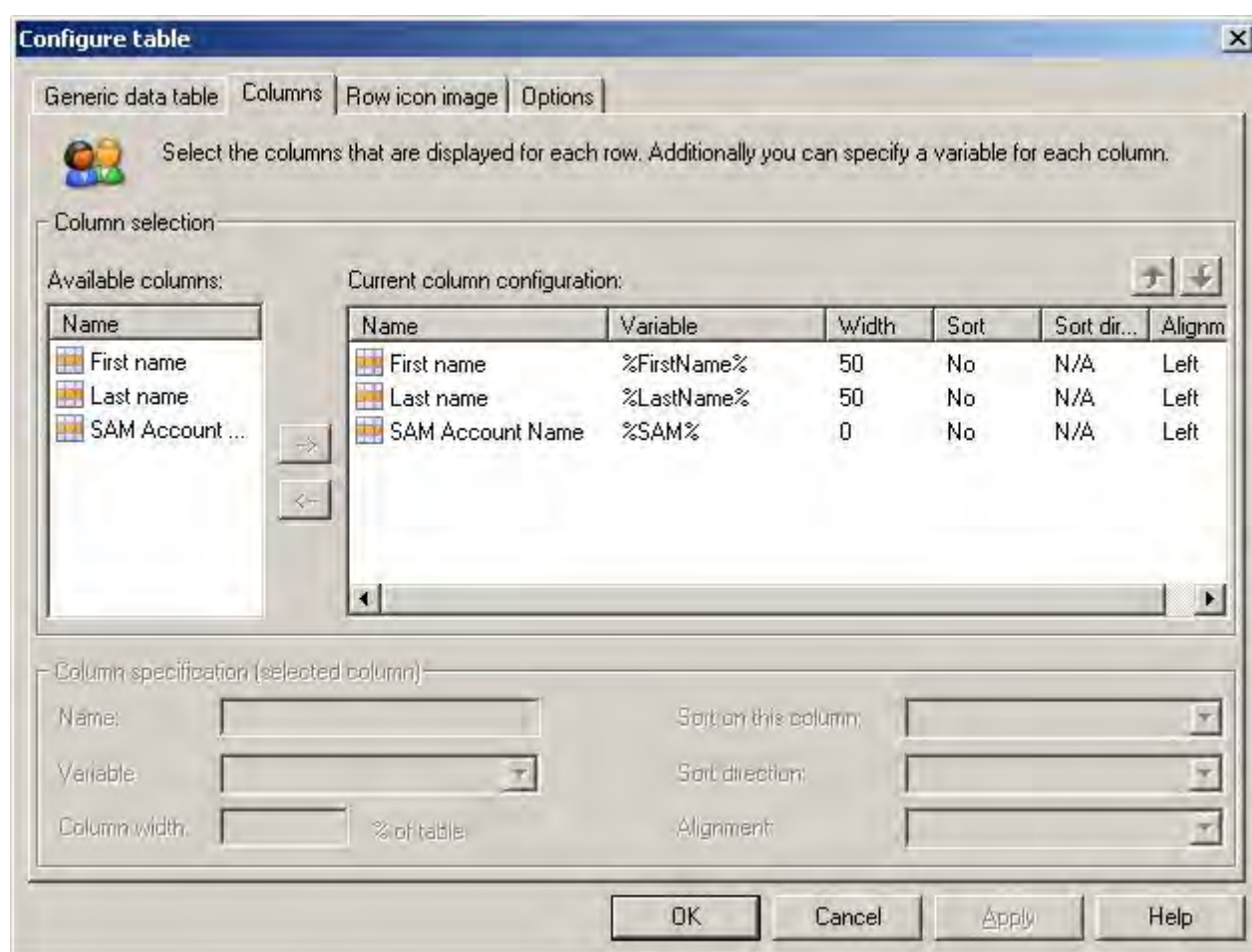


Figure 43: Testing the LDAP query

7. Finally, click **OK**.

## Step 2 - Building the form script

The project script for the **Reset Password** project should perform the following tasks:

- **Obtain the user account of the selected user** - this will be done using the **Get User (AD)** script action. With the **domain name** and the **sAMAccountname** of the selected user as input, this script action can automatically obtain the user's logon account in Active Directory. This result is stored in the variable **%UserObject%**.
- **Set the password of the selected user** - Using the script action **Edit user logon**, the user account stored in **%UserObject%** can be edited. The script action generates a new password and updates the user logon accordingly.
- **Reporting status information to the delegated user** - in the **UMRA Forms** client, the project execution results are not displayed in a log window as in the **UMRA Console**. For a live project, you could consider extending the project with a form showing the end user a summary status report.



Figure 38 - Extending the form with a summary status report

For the purpose of this example project, we will simply display a small text message window showing the new password of the selected user.



Figure 39 - Text message showing the new password for the selected user

## Obtaining the user account of the selected user

To specify a user account, there are three different methods:

- Specify the full **LDAP name**
- Specify **Domain name + OU-Container + Fullname**

- Specify **Domain + UserName**. In this last case, UMRA will automatically try to find the full LDAP name.

For our project, we will be using the third method. The **UserName** is identical to the **sAMAccountname (%SAM%)**.

1. Click the **Actions** tab in the **Actions-Network-Form fields** window. Open the **User→Active Directory** folder and drag the **Get User (AD)** script action to the project script window.
2. Set the properties for this script action as shown in the following table.

Property value	Set to	Description
Domain	<b>MY-DOMAIN</b>	Replace <b>MY-DOMAIN</b> with the name of your domain.
Organizational Unit-Container	Do not specify	
Full name	Do not specify	
Username	<b>%SAM%</b>	This variable is linked to the LDAP table column containing the <b>sAMAccountname</b> .
LDAP name	Do not specify	
Domain controller	Do not specify (Active Directory will choose one automatically through serverless binding)	
User Object	Output variable only ( <b>%UserObject%</b> )	

#### Resetting the password of the selected user

To edit the user account obtained with the **Get User (AD)** script action, the script action **Edit user login** is used.

1. Click the **Actions** tab in the **Actions-Network-Form fields** window. Open the **User→Active Directory** folder and drag the **Edit user login** script action to the project script window.
2. Specify the script action property values as shown in the table below.

Property value	Set to
User Object	<b>%UserObject%</b> - this is the output variable obtained from the <b>Get User (AD)</b> script action.
Username	Do not specify
Domain	<b>%Domain%</b>
Password generator	Choose the default password generation option.



Password	%Password% - This variable will contain the output value from the <b>Password generator</b> property.
----------	---

#### Displaying the project results in a text message window for the delegated user

1. Click the **Actions** tab in the **Actions-Network-Form** fields window. Open the **Variable actions**→**Variable operations** folder and drag the **Set variable** script action to the project script window. Make sure it is positioned as the last script action in your project script.
2. Doubleclick in the lower section of the project script window to open the **Properties** window.
3. In the **Variable name** list, enter the variable **%ScriptMessage%**. This variable will hold the status information for the selected user.

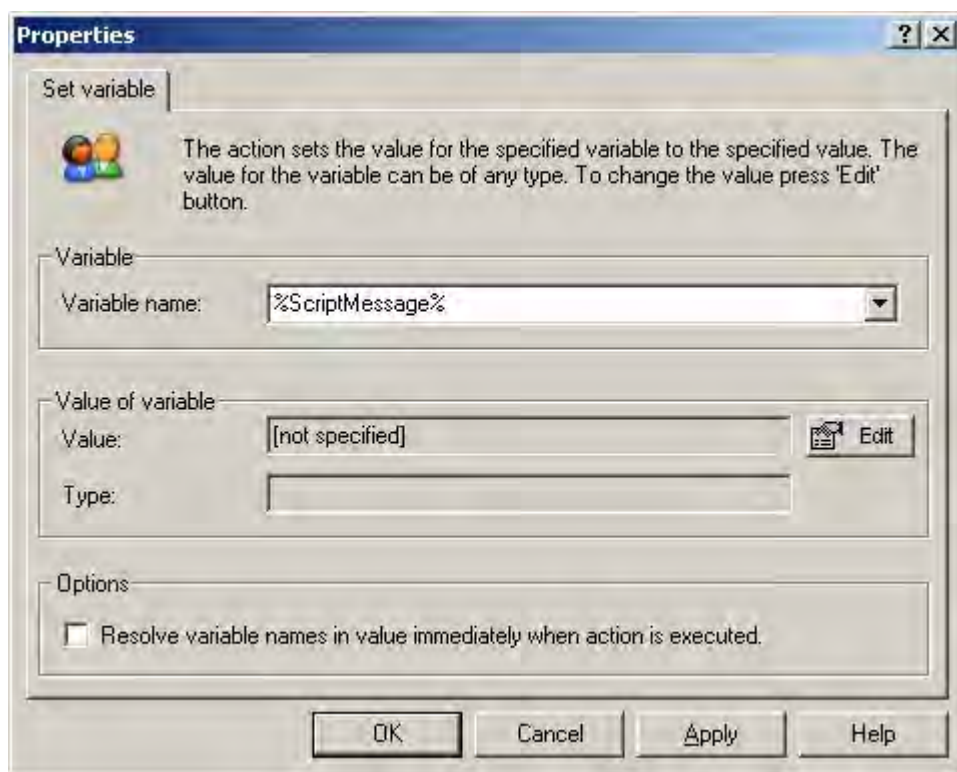
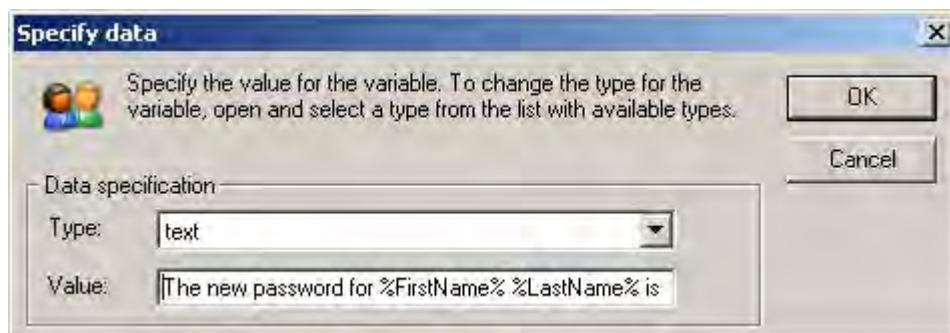


Figure 40 - Specifying the variable to hold status information

4. Click the **Edit** button in the **Value of variable** section. This will bring up the **Specify data** window.



5. In the **Value** field, enter the text “**The new password for %FirstName% %LastName% is %Password%**”. You will remember that you have linked the First name and Last name columns of the LDAP table to these variables in section Linking a table column to a variable. When the script action is executed for the selected user, these variables will therefore be replaced with their actual values.

### Step 3 - Specifying security

Finally, you need to specify who will be allowed to access and run the form in the delegation client (**UMRA Forms**).

1. Right-click the form and choose **Form properties**.



2. Click the **Security** tab in the **Configure form properties** window.

In this window, you can add the users and groups who are allowed to access and run the form. Since we want to delegate the form to non-admins, we will add the **Helpdesk** group. Please select a user or group here which exists in your own organization.

3. Click the **Add** button to specify an existing account name.

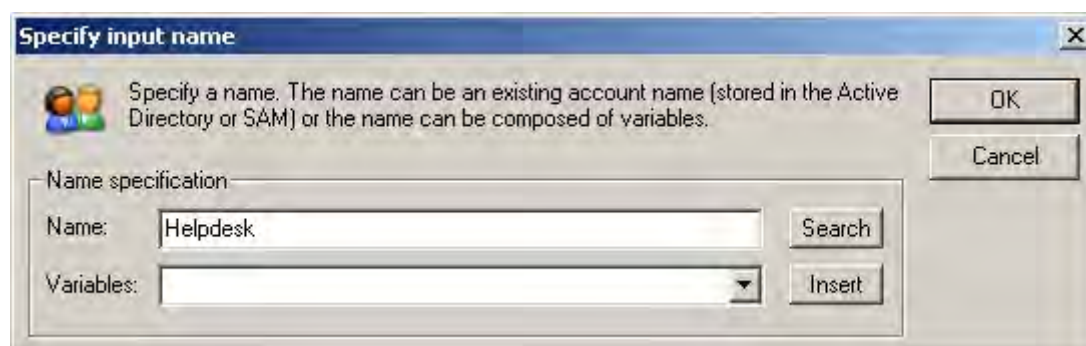


Figure 41 - Specifying the users allowed to access and run the form



4. Click **OK**.

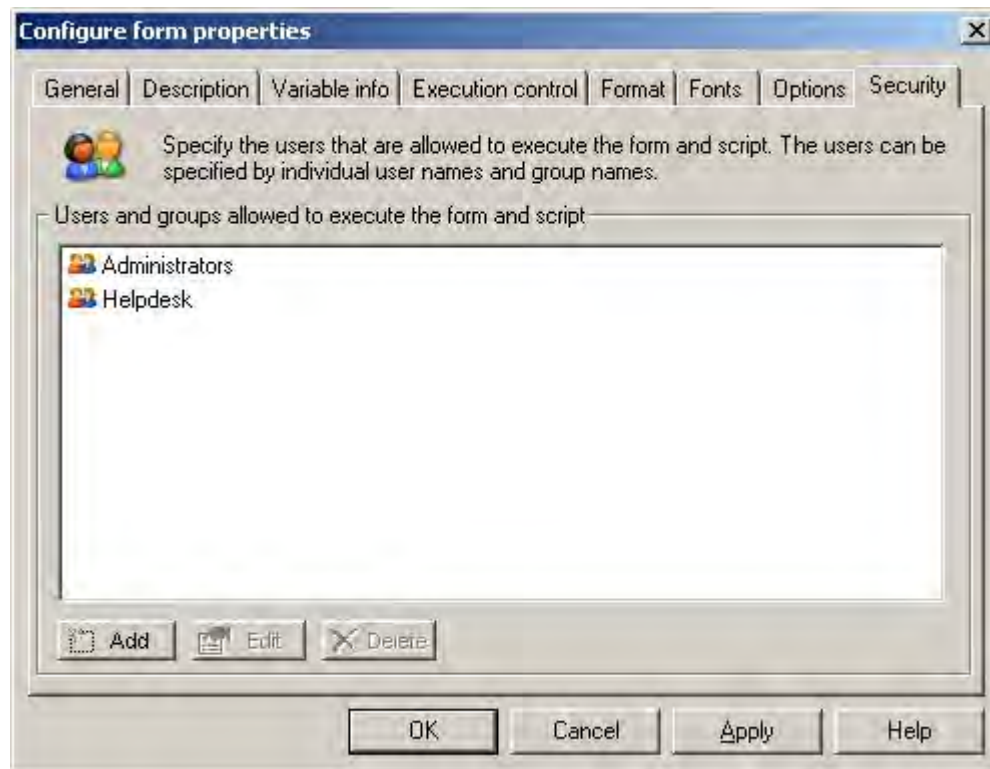



Figure 42 - The Helpdesk group has been granted privileges to execute the form

### Testing the project script

In the **UMRA Console**, a detailed log is generated with the results of each and every executed script action, date and time of script execution, properties set, encountered errors, etc.

1. First of all, make sure that you are running UMRA in **Test only** mode (you should see the icon



. If you see the icon  instead, switch to test mode by clicking the icon).

Select the **Preview** tab of your project and select a user from the table.



Figure 43 - Testing the delegation project

- Click the **Reset Password** button. The project script is now executed using the selected data. A text window is displayed with the project execution result.



Figure 44 - Text message showing reset password result

If you now look at the **Log messages** window, you will notice that all the project script execution details are listed. The text in **bold** has been added to explain the individual sections.

**Log start showing the build version and the date and time when the script was executed:**

Starting User Management Resource Administrator session, build 1233 at 15:51:50 03/15/2006

**Message showing that the script is run in test mode:**

15:51:50 03/15/2006 \*\*\*\*\* TEST ONLY \*\*\* STARTING JOB SIMULATION \*\*\* TEST ONLY \*\*\*\*\*

15:51:50 03/15/2006 Submit form to UMRA service on computer 'tools4ev-opa0zi' (test only).

**This section provides an overview of the variables which have been set for the project:**

15:51:50 03/15/2006 Variable 1: %FirstName%=Thomas

15:51:50 03/15/2006 Variable 2: %LastName%=Franklin  
15:51:50 03/15/2006 Variable 3: %SAM%=franklint  
15:51:50 03/15/2006 Variable 4: %UmraFormSubmitAccount%=T4EDOC\E. van Wezel  
15:51:50 03/15/2006 Variable 5: %NowDay%=15  
15:51:50 03/15/2006 Variable 6: %NowMonth%=03  
15:51:50 03/15/2006 Variable 7: %NowYear%=2006  
15:51:50 03/15/2006 Variable 8: %NowHour%=15  
15:51:50 03/15/2006 Variable 9: %NowMinute%=51  
15:51:50 03/15/2006 Variable 10: %NowSecond%=50

**Script action Get User (AD) is executed to find the user account for the selected user:**

15:51:51 03/15/2006 Finding AD user 't4edoc\franklint' (Domain\Username). LDAP name 'CN=Thomas Franklin,CN=Users,DC=t4edoc,DC=local'.

**The user account is found:**

15:51:51 03/15/2006 User account 'LDAP://CN=Thomas Franklin,CN=Users,DC=t4edoc,DC=local' successfully found in Active Directory.

**Script action Edit user logon is executed:**

15:51:51 03/15/2006 Editing logon properties of user account 'LDAP://CN=Thomas Franklin,CN=Users,DC=t4edoc,DC=local'. User account specified by 'User Object'.

**The obtained user account is updated with a newly generated password:**

15:51:51 03/15/2006 Editing logon properties of user account 'LDAP://CN=Thomas Franklin,CN=Users,DC=t4edoc,DC=local'. Password generated ('Strong, 7 chars with 1 numeric, 1 special (variable: %Password%)') and set in variable '%Password%'.

**Line indicating who executed the project form and when:**


15:51:51 03/15/2006 Form message: '03/15/2006,15:51:50,"T4EDOC\E. van Wezel","Form submit",OK,"Sample Form Project - Reset Password2"'

End of session


### 3.2.9. Appendix A - Script actions

The table below shows an overview of several script actions available in UMRA. Together, these script actions cover virtually every aspect of user account management. See the UMRA Reference Guide in the online Help for a complete list and detailed information about the script actions.

#### Active Directory User Actions

 *Script Action: Create User (AD) on page 3*


 *Script Action: Create contact (AD) on page 21*

 *Script Action: Get user (AD) on page 31*

 *Script Action: Edit user (AD) on page 37*

 *Script Action: Edit user logon (AD) on page 47*


 *Script Action: Get user table (AD) on page 51*

 *Script Action: Delete user (AD) on page 55*


 *Script Action: Set User Group Memberships (AD) on page 56*

 *Script Action: Remove user group memberships (AD) on page 60*

 *Script Action: Move - rename user (AD) on page 63*


 *Script Action: Move cross-domain (AD) on page 67*

 *Script Action: Create Exchange Mailbox (2000/2003) on page 156*

 *Script Action: Edit Exchange mailbox (2000/2003) on page 159*

 *Script Action: Modify Exchange mailbox permissions (2000/2003) on page 162*

 *Script Action: Move Exchange mailbox on page 167*


 *Script Action: Delete Exchange mailbox (2000/2003) on page 168*

 *Script Action: Manage Exchange recipient mail addresses (2003/2000) on page 169*


#### Non Active Directory

 *Script Action: Create User (no AD) on page 68*


 *Script Action: Edit user (no AD) on page 79*

 *Script Action: Delete user (no AD) on page 86*

 *Script Action: Set User Global Group Memberships on page 88*


 *Script Action: Add account to local group on page 90*

 *Script Action: Remove group member on page 94*

 *Script Action: Set primary group (non AD) on page 155*

### General user actions

 *Script Action: Edit user logon (AD) on page 47*


 *Script Action: Get user info on page 101*

 *Script action: Terminal Services user settings on page 104*


 *Script action: Get terminal services user settings on page 111*

 *Script Action: Dial-in user settings on page 114*


### Active Directory


 *Script action: Create object (AD) on page 117*

 *Script Action: Delete Object (AD) on page 119*

 *Script Action: Get attribute (AD) on page 120*

 *Script Action: Set attribute (AD) on page 124*


 *Script Action: Delete attribute value (AD) on page 129*


 *Script Action: Set group membership (AD) on page 135*


 *Script Action: Remove specific group memberships (AD) on page 137*


 *Script Action: Create group (AD) on page 138*

 *Script Action: Get object (AD) on page 145*

 *Script Action: Search object (AD) on page 146*

 *Script Action: Move cross-domain on page 67*


 *Script action: Get primary group on page 154*

 *Script Action: Set primary group (AD) on page 155*

## File system

 *Script Action: Create Directory on page 341*


 *Script Action: Get file/directory info on page 345*


 *Script Action: Copy directory on page 347*

 *Script Action: Rename file or directory on page 352*

 *Script Action: Setup Security on page 354*

 *Script Action: Delete directory on page 357*

 *Script Action: Create share on page 361*

 *Script Action: Edit share on page 364*


 *Script Action: Delete share on page 366*

## Other actions

 *Script Action: Execute Command Line on page 369*


## Services

 *Script action: List services status on page 373*

 *Script Action: Execute service command on page 377*

 *Script Action: Configure service on page 379*


## Printer

 *Script Action: List printer documents on page 380*


 *Script Action: Execute print job command on page 382*


## LDAP

 *Script Action: Setup LDAP session on page 383*

 *Script Action: Load LDAP modification data on page 386*


 *Script Action: Add directory service object (LDAP) on page 387*

 *Script Action: Modify directory service object (LDAP) on page 388*

 *Script Action: Delete directory service object (LDAP) on page 389*


 *Script Action: Search LDAP on page 391*

## Table


 *Script Action: Generate generic table on page 527*

 *Script Action: Manage table data on page 528*


## Database

 *Script Action: Update database - introduction on page 539*


 *Script Action: Update database - Database (on page 538)*

 *Script Action: Update database - SQL Statements on page 539*


 *Script Action: Update database - Variable list (see "Variable list" on page 778)*

 *Script Action: Update database - Test (on page 541)*

## Name generation

 *Script Action: Generate name(s) on page 542*

## Variable operations

 *Script Action: Set Variable on page 544*

 *Script Action: Set encrypted variable on page 546*

 *Script Action: Split Variable on page 546*


 *Script Action: Format Variable Value on page 549*

 *Script Action: Update numeric variable on page 550*

 *Script Action: Update date-time variable on page 552*

 *Script Action: Convert value of variable on page 554*

 *Script Action: Convert text to date/time on page 555*

 *Script Action: Convert to multi-valuevariable on page 556*


 *Script Action: Manage multi-text value variable on page 558*

 *Script Action: Merge multi-text variable values on page 559*

 *Script Action: Export Variables on page 559*

 *Script Action: Delete variable on page 562*

 *Script Action: Encrypt text on page 563*


 *Script Action: Generate random number on page 564*


 *Script Action: Generate password on page 565*

 *Script Action: Log Variables on page 565*


## Programming

 *Script Action: Map Variable on page 567*

 *Script Action: Go to Label on page 569*

 *Script Action: If-Then-Else on page 570*

 *Script Action: Execute script on page 571*

 *Script Action: For-Each on page 572*

 *Script Action: Delay on page 574*

 *Script Action: No operation on page 575*

## Mail

 *Script Action: Send mail message on page 575*



### 3.2.10. Appendix B - Installing the UMRA Service

The service can be installed on any server running Windows 2000/2003. It is advised however, to install the **UMRA service** on a computer that is a member of the domain containing the user accounts and computer services you wish to manage. For testing purposes, you can also install the **UMRA service** on the same computer that runs the **UMRA Console** application.

The **UMRA Service** is completely controlled by the **UMRA Console** and can be set up by completing the following steps:

1. Start the **UMRA Console** application.
2. Select **UMRA service** → **Install or upgrade service**. This will launch the **UMRA service wizard**.
3. Select the option **Install or upgrade the UMRA service** and click **Next**.



Figure 45 - Introduction screen of the UMRA service installation wizard

4. Enter the name of the server where you want to install the **UMRA service**. This can also be a local computer.

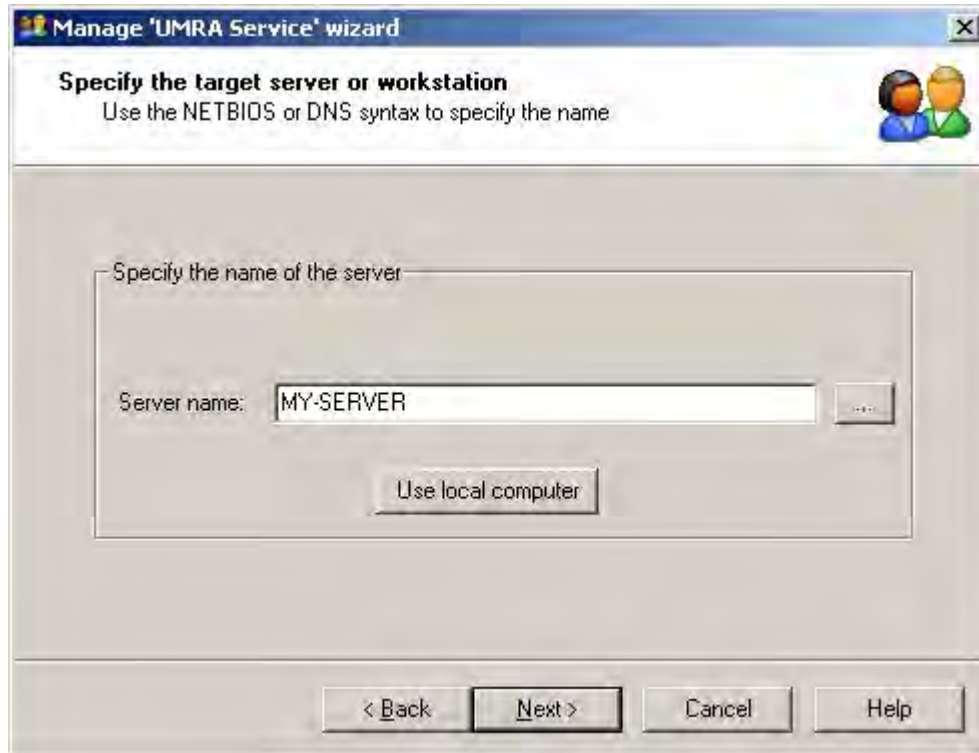


Figure 46 - Specifying the computer where the UMRA service should be installed

5. Specify the TCP/IP Port and click **Next**. The **UMRA service** communicates with the **UMRA Console** and **UMRA Forms** client using the TCP/IP protocol. For this communication a port must be specified. By default, port number 56814 is used, but you can specify any other available port.

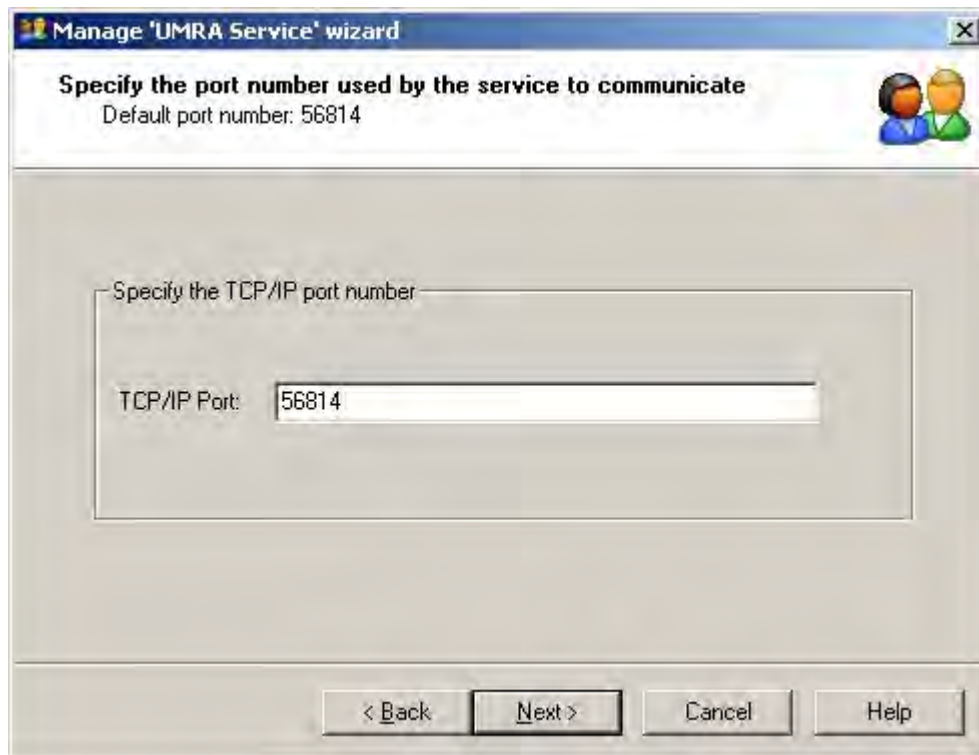


Figure 47 - Specifying the TCP/IP port number for the UMRA service

6. Before the **UMRA service** is installed on the specified computer, the **UMRA service** executable files are copied to the specified computer. Please specify the directory where these files need to be copied.



Figure 48 - Specifying the directory where the installation files should be copied to

7. Specify the user account for the **UMRA service** and click **Next**.

The **UMRA service** uses a Windows 2003/2000 account to run. Since all project scripts are executed by the **UMRA service**, you have to ensure that this account has sufficient administrative privileges. By default, the **UMRA service** installation wizard specifies the name **DOMAIN\UmraSvcAccount** for the service account. Also, a random strong password is generated. This password is not known to anyone. If the account does not exist, the account is created with the generated password.



Figure 49 - Specifying an account for the UMRA service

8. Specify the access rights for the **UMRA service** account and click **Next**.

The **UMRA service** must have sufficient administrative privileges to execute its tasks. These privileges are determined by the service account used by the **UMRA service**. By adding the service account to one or more administrative groups, the account can be granted sufficient access rights.

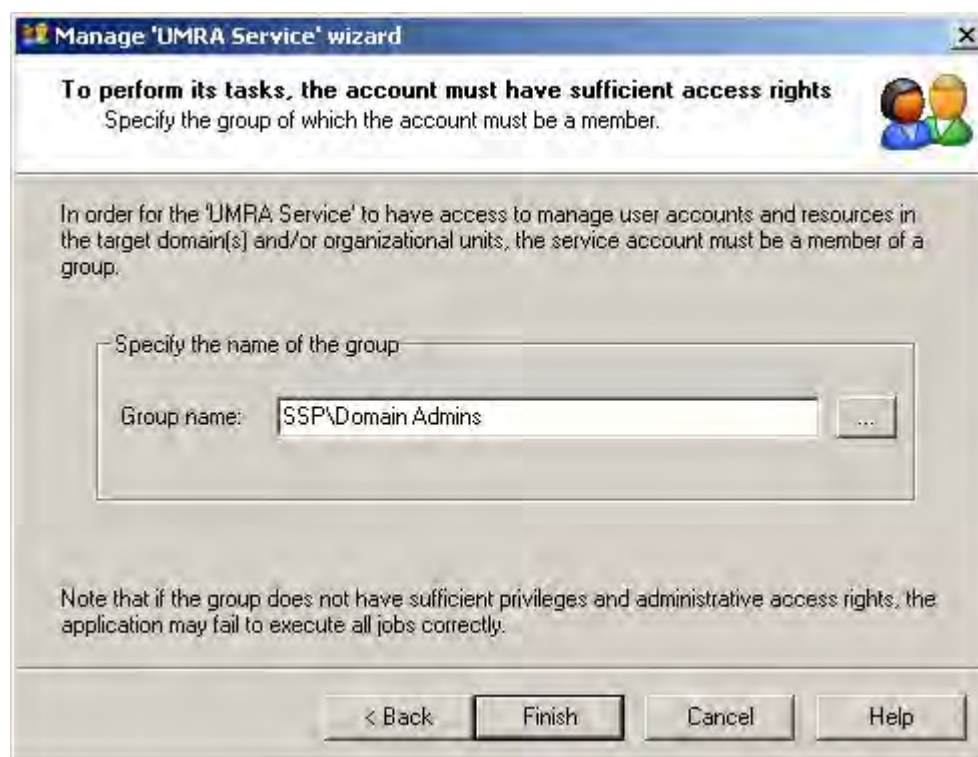


Figure 50 - Setting privileges for the UMRA service account

9. Click **Finish** to complete the **UMRA service** installation.

In case you have already installed the **UMRA Service** during the installation of the program files, please follow the steps below:

1. Start the **UMRA Console** application.
2. Connect to the **UMRA service** by selecting **UMRA Service** → **Connect** and connect to the computer on which the **UMRA service** is installed.

---

### 3.3. Integrate UMRA with other applications using COM

#### Component Object Model (COM)

This document describes how Microsoft's Component Object Model (COM) is used in combination with User Management Resource Administrator (UMRA). Using COM with UMRA is referred to as **UMRA COM**.

#### ASP and IIS

One of the many applications which can utilize UMRA COM can be used is Microsoft's Internet Information Services (IIS). In web applications running on IIS, Active Server Pages (ASP) may contain UMRA COM objects to interface with the **UMRA Service**. A substantial part of this document focuses on UMRA COM objects used in ASP pages.

There is a lot of documentation available on both COM and ASP. This document only briefly describes the backgrounds of COM and ASP. A number of references to these subjects are listed in at the end of this document.

UMRA COM is part of the User Management Resource Administrator Automation module (UMRA Automation). To use **UMRA COM**, an **UMRA Automation** license is required.



*Read the full PDF version of UMRA COM*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/Using-UMRA-with-COM-objects.pdf>

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*



### **3.3.1. UMRA COM**

This section explains what UMRA COM is and what its main functions are .

#### **Component Object Model (COM)**

The Component Object Model (COM) is a technology that is used by applications to interact with other applications. The COM technology is designed by Microsoft. The most important Microsoft applications that use COM are:

- Internet Information Services
- Office applications
- Visual Basic Scripting (VB) and Visual Basic

#### **COM objects and interfaces**

In principle, a COM object is a piece of software that implements one or more functions. Different COM objects support different functions. The functions of a COM object are accessible by means of the interface of the COM object. The COM object itself is regarded as a black box that implements one or more functions accessible through its interfaces.

Applications that support COM can create COM objects. By accessing the interface functions of the COM object, the functions of the COM object are executed by the calling application. The syntax to create COM objects and access the interface functions of a COM object is extremely general: this is the main reason why COM objects can be used by so many different applications: The same COM object can be created and used in ASP-pages, Word documents and Visual Basic scripts.

All applications that support COM use some kind of programming or script language to implement COM. The procedure used is always the same:

1. The COM object is created;
2. The interface functions of the COM object are accessed;
3. Returned variables can be processed in the application.

An application can use multiple COM objects and COM objects can use other COM objects.

#### **COM registration**

In order for an application to use a COM object and the interface functions of the object, the COM object must be registered. Once a COM object is registered, all applications that support COM can use the COM object. In most cases, COM object are registered automatically.

#### **Type library**

In most cases, the COM object code is contained in a file with the .DLL extension. Such a file contains all code needed to use the COM objects it implements. Besides the COM objects, the file can also contain a

so called type library that describes the COM objects and the interfaces that are used to access the COM objects.

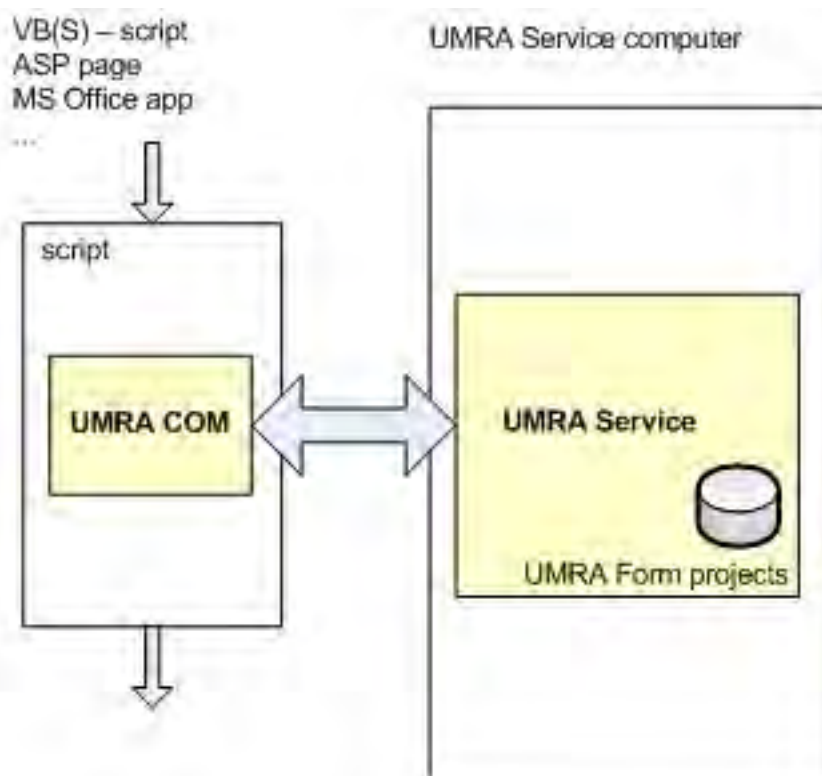
### UMRA COM - Main functions

User Management Resource Administrator contains several COM objects as part of the **UMRA Automation** module. The main functions of UMRA COM are:

- Connect to an **UMRA Service** and execute an **UMRA project** on the **UMRA Service**;
- Read and process the information returned by the project variables.

### UMRACOM.DLL

All UMRA COM objects are contained in a single file: UMRACOM.DLL. The COM objects are automatically registered on the computer when the **UMRA Console** or **UMRA Service** application is installed.



*Figure 1: UMRA COM objects used in a script accessing the UMRA Service.*

Any (form) project that is maintained by the **UMRA Service** can be accessed using UMRA COM.

---

### 3.3.2. UMRA COM objects

This section provides an introduction to using UMRA COM objects.

#### Introduction

#### UMRA COM objects

UMRA COM supports four types of COM objects. The main functions of the UMRA COM objects are to:

- access the **UMRA Service**,
- execute UMRA projects on the **UMRA Service**
- manage the data retrieved from the **UMRA Service**.

UMRA COM object	Description
Umra	The main UMRA COM object. Used to connect to the UMRA Service, execute an UMRA Form project on the UMRA Service and manage variables, values and form project tables.
UmraFormProject	Intermediate COM object used to access the data of a form table.
UmraFormTable	COM object used to access the data of a form project table.
UmraDataTable	COM object used to access the data of table variable.

*Table 1: The UMRA COM objects.*

#### UMRA COM software

The COM objects are contained in a single DLL: **UMRACOM.DLL**. This DLL is part of the **UMRA Automation** module and installed with the **UMRA Console** application. The COM objects are registered automatically. Part of the UMRA COM software is the type library that describes the COM objects and interfaces of UMRA COM.

#### Using UMRA COM

The UMRA COM objects can be used in many environments. The most important functions implemented with UMRA COM objects are the following three scenarios:

1. Executing a UMRA project on the **UMRA Service** and processing the results.

2. Accessing the data of a form table contained in the form of an UMRA Form project.
3. Accessing the data of a table variable generated by a UMRA project script.

#### **Executing an UMRA project on the UMRA Service**

The procedure to execute a UMRA project on the UMRA Service is always the same. The project must be setup using the **UMRA Console** application. The project type must be a form project. The project only needs to contain a script, not a form. A form project with a form can also be used. The security settings of the project must be set so that the project can be accessed from the security context of the UMRA COM objects accessing the **UMRA Service**. In general, the UMRA COM objects are executed with the security context of the user account that is logged on to the computer that runs the application that uses the UMRA COM objects.

#### **Executing a project on the UMRA Service**

Procedure to execute the script of the UMRA project on the UMRA Service using UMRA COM:

1. An instance of the **Umra COM** object is created.
2. The **Umra.Connect** method is used to connect to the UMRA Service.
3. The list with variables maintained by the **Umra COM** object is created. All variables that are required to execute the script of the UMRA project must be initialized. For each variable, the name of the variable and the value must be specified. Normally, the values are taken from the application that uses the **UMRA COM** objects. Several methods of the **Umra COM** object are available to setup the list with variables.
4. The method **Umra.ExecuteProjectScript** is called to request the UMRA Service to execute the script of the specified project. On return, the variables updated by the script are stored in the list with variables.
5. The list with variables can be accessed to process the returned information.

### Accessing the data of a form table

UMRA Form projects can contain a form. The form can contain a table presented with UMRA Forms. The table data can be access using UMRA COM using the following procedure:

1. An instance of the following UMRA COM objects must be created: **Umra**, **UmraFormProject**, **UmraFormTable**.
2. The **Umra.Connect** method is used to connect to the UMRA Service.
3. The method **Umra.LoadFormProject** is used to load a form project in the UMRA COM object. As one of the arguments, the **UmraFormProject** object is passed. On success, the passed COM object **UmraFormProject** contains the form of the project, including the form tables that are part of the form.
4. The method **UmraFormProject.GetFormTable** is called to load the **UmraFormTable** object. The object **UmraFormTable** is one of the arguments of the method.
5. On success, the COM object **UmraFormTable** contains the table data. The method **UmraFormTable.GetCellText** can be used to access the table data.

### Accessing the data of a table variable

UMRA supports many script actions that generate table data. Such a table is normally stored in a single variable and the action is part of a UMRA project script. When the project is executed using UMRA COM, the variable is returned. The following procedure describes how to access the data of the table variable:

1. Instances of the UMRA COM objects **Umra** and **UmraDataTable** are created.
2. The UMRA COM object **Umra** is connected to the UMRA Service and a project is executed. See **Execute an UMRA project on the UMRA Service** for more information. The list with variables maintained by the **Umra** COM object now contains a variable with table data.
3. The member **Umra.GetVariableDataTable** is called to load the table data. The method requires two arguments: The name of the variable and the **UmraDataTable** object. On success, the **UmraDataTable** contains the table data of the variable.

4. The method **UmraDataTable.GetCellText** is used to access the data of the individual table cells.

#### **UMRA COM object reference**

The **UMRA COM object** exposes the following interfaces:

**Umra** : General interface to setup the connection with the UMRA Service, execute project, manage variable lists.

**UmraFormProject** : Simple interface to obtain access to form project tables.

**UmraFormTable** : Simple interface to read data from a form project table.

**UmraDataTable** : General interface to manage tables

#### *Umra*

#### **UMRA COM object: Umra**

This is the main UMRA COM object. The object is used to connect to an **UMRA Service**, execute a project on the **UMRA Service** and retrieve data from the **UMRA Service**.

ClearVariables

#### **Umra.ClearVariables**

```
void ClearVariables()
```

Argument	Type	Description
no arguments..		

The interface method deletes all variable-value pairs from the list with variables maintained by the **UMRA COM** object. When completed, the list is empty.

Connect

**Umra.Connect**

**long Connect([in] BSTR Host, [in] long PortNumber)**

Argument	Type	Description
Host	in	The name of the host that runs the <b>UMRA Service</b> to which the COM object must connect. The name can be specified as DNS or NETBIOS name.
PortNumber	in	The port of host running the <b>UMRA Service</b> . The default port is 56814.

The interface method connects to an **UMRA Service**. The name of the computer running the **UMRA Service** and the port number must be specified. Normally, this is the first method called when the **UMRA COM** object is created.

When the connection is established, the connection parameters are stored within the object. The COM object can be connected to only one **UMRA Service** at a time. The return value should be zero on success. If the return value is not zero, the COM object is not connected to the **UMRA Service**.

ExecuteProjectScript

**Umra.ExecuteProjectScript**

**long ExecuteProjectScript([in] BSTR ProjectName)**

Argument	Type	Description
ProjectName	In	The name of the project of which the script must be executed by the <b>UMRA Service</b> .

The interface method requests the **UMRA Service** to execute the script

of the specified project immediately. The COM object must be connected to the **UMRA Service** (see `Umra.Connect`) and the project must be maintained by the **UMRA Service**. The current list with variable-value pairs is used to execute the script of the project.

When the project is executed successfully, the return value is zero. In this case, the list with variables is updated as defined in the UMRA project script. The returned variables can be used for further processing. The log information in the COM object is updated. The method `Umra.GetLogMsg` can be used to retrieve the log information.

`GetConnectionInfo`

**Umra.GetConnectionInfo**

**long GetConnectionInfo(**

**[out] VARIANT \* ServerName,**  
**[out] unsigned long \* RpcPortNumber,**  
**[out] unsigned long \* RpcHandle)**

Argument	Type	Description
ServerName	out	The name of computer (host) to which the COM object is connected.
RpcPortNumber	out	The port number used on the connected host.
RpcHandle	out	An internal identifier used by the COM object to communicate with the <b>UMRA Service</b> .

The interface method retrieves connection information from the **UMRA COM** object. The name of the host, port number and an internal identifier is retrieved. On success, the return value is zero. If the COM object is not connected to an **UMRA Service**, the method returns a non-zero value.



GetConnectionString

**Umra.GetConnectionString**

**long GetConnectionString(**

**[out] VARIANT \* ConnectionString)**

Argument	Type	Description
ConnectionString	out	An encrypted text string that represents the current connection between the UMRA COM object and the UMRA Service. The text can be used later to reconnect to the same UMRA Service, using the same <i>UMRA Session</i> on page 81.

The interface method retrieves connection information from the **UMRA COM** object. The information contains the name of the host, port number and the UMRA session identification. The connection string can be used later, for instance in another ASP or ASPX page, to reconnect to the same UMRA Service, using the same *UMRA session* on page 81.

GetHostName

**Umra.GetHostName**

**long GetHostName([out] VARIANT \* HostName)**

Argument	Type	Description
HostName	Out	The name of the host to which the <b>UMRA Service</b> is connected.

The interface method retrieves the name of the host to which the **UMRA COM** object is connected. The name corresponds with specified host of method **Connect**.

The return value should be zero on success. If the return value is not zero, the COM object is not connected to the **UMRA Service** and no name is returned.

GetLogMsg

**Umra.GetLogMsg**

**long GetLogMsg([out] VARIANT \* Msg)**

Argument	Type	Description
Msg	Out	The log information stored by the COM object.

The interface method retrieves the log information stored by the interface. The log information is refreshed when the interface communicates with the **UMRA Service**. The log information describes the success or failure of the request that was last executed. When a new method is called that involves communication with the **UMRA Service**, the contents are always reset first.

GetLogMsgCount

**Umra.GetLogMsgCount**

**long GetLogMsgCount([out] VARIANT \* Count)**

Argument	Type	Description
Count	Out	The number of messages in the Log of the Com Object.

New in Version 10.7 of UMRA

The interface method retrieves the current number of log messages in the log information stored by the interface. The log information is

refreshed when the interface communicates with the **UMRA Service**. The log information describes the success or failure of the request that was last executed. When a new method is called that involves communication with the **UMRA Service**, the contents are always reset first.

This number can be used as an indication of the valid range for the GetLogMsgEx method.

GetLogMsgEx

**Umra.GetLogMsgEx**

**C++**

```
long GetLogMsgEx([in] ULONG Index,[out] VARIANT*
MessageTime,[out] VARIANT* MessageMask, [out] VARIANT*
MessageLogCode, [out] VARIANT * MessageString, [out,retval] LONG*
RetVal)
```

**VB**

```
RetVal = GetLogMsgEx([in] ULONG Index,[out] VARIANT*
MessageTime,[out] VARIANT* MessageMask, [out] VARIANT*
MessageLogCode, [out] VARIANT * MessageString)
```

Argument	Type	Description
Index	In	The index of the Message record from the log to be retrieved. range [0,Count-1]
MessageTime	out [VT_I4]	The raw time as stored in the log (32 bit time_t), (LONG)

MessageMask,	out [VT_UI4]	The LogMask (ULONG). Indication of severity
MessageLogCode,	out [VT_I4]	The Errorcode associated with the message
MessageString	out [VT_BSTR]	The log information stored by the COM object.
RetVal	out,retval	Returnvalue of the function. 0 on success

New in version 10.7 of UMRA

#### C++

Returns COM error code when COM error occurs, 0 otherwise.

RetVal contains the result reported by the called function. Retval is 0 in case of success.

#### VB and VBSCRIPT

RetVal contains the result reported by the called function. Retval is 0 in case of success.

The interface method retrieves a single log record from the log information stored by the interface. The log information is refreshed when the interface communicates with the **UMRA Service**. The log information describes the success or failure of the request that was last executed. When a new method is called that involves communication with the **UMRA Service**, the contents are always reset first.

To determine the possible range of the records to retrieve, use the GetLogMsgCount method immediately prior to a call to this method.

GetPortNumber

**Umra.GetPortNumber**

**long GetHostPortNumber([out] long \* PortNumber)**

Argument	Type	Description
PortNumber	Out	The number of the port to which the COM object is connected.

The interface method retrieves the port number of the host to which the **UMRA COM** object is connected. The number corresponds with specified port number of method **Connect**.

The return value should is zero on success. If the return value is not zero, the COM object is not connected to the **UMRA Service** and no port number is returned.

GetScriptExecutionInfo

**Umra.GetScriptExecutionInfo**

**long GetScriptExecutionInfo(**

**[out] long \* ScriptErrorCount,**

**[out] VARIANT \* ScriptMessage)**

Argument	Type	Description
ScriptErrorCount	Out	The number of errors occurred executing the script of the project on the <b>UMRA Service</b> ..
ScriptMessage	Out	The log information generated by the <b>UMRA Service</b> when the last project was executed.

The interface method retrieves the logging and error count information from the last project executed through the COM object on the **UMRA Service**. The number of errors that occurred when the script was

executed is returned and the log information generated by the **UMRA Service**.

On success, the method returns zero. If the information cannot be obtained from the COM object, a non-zero value is returned.

GetVariableInfo

#### **Umra.GetVariableInfo**

**long GetVariableInfo( [in] BSTR VariableName, [out] VARIANT\* VariableInfo)**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
<b>VariableName</b>	in	The name of the variable of which the Information must be retrieved.
<b>VariableInfo</b>	out[VT_I4]	The Datatype of the Variable. The number has the following meaning 0 unknown or unspecified type. 1 Text 12 Table  Any other value corresponds to an UMRA-Specific internaldatatype.

#### **Purpose**

Use this function to determine which of the GetVariable methods should be used to retrieve the value

If **VariableInfo** equals 12, use **GetVariableDataTable** to retrieve the data.

if **VariableInfo** equals 1, use **GetVariableText** to retrieve the data.

For any other type, use **GetVariableText** to get a textual representation of the value.

GetVariableDataTable

**Umra.GetVariableDataTable**

**long GetVariableDataTable([in] BSTR \* VariableName, [in] IUnknown \* pDataTable)**

Argument	Type	Description
VariableName	In	The name of the variable that holds the table data to be retrieved.
pDataTable	In/out	The retrieved table data. The argument must be specified as one of the other UMRA COM object types: UmraDataTable. This COM object can then be used to access the data of the table.

The interface method retrieves the data of a table variable. When an **UMRA Form project** generates a variable with table data, the variable and table data is returned in the list with variables maintained by the **UMRA COM** object. To access the table data, a special **UMRA COM** object is used: **UmraDataTable**. An instance of such an COM object must be passed as one of the arguments of this method. On success, the table data is copied from the variable into the passed COM object. The interface methods of the **UmraDataTable** object can then be used to access the table data.

On success, the method returns zero. If the variable is not found in the list with variables, or when the variable does not contains table data, a non-zero value is returned.

GetVariableText

**Umra.GetVariableText**

**long GetVariableText( [in] BSTR VariableName, [out] VARIANT\* ValueText)**

Argument	Type	Description
VariableName	in	The name of the variable of which the textual value must be retrieved.
ValueText	out	The text value of the specified variable.

The interface method retrieves the text value of a specified variable. The variable must be contained by the list of variables maintained by the COM object. The method is normally used when a project is executed on the **UMRA Service** and variable values are generated by the UMRA project script.

On success, the return value is zero. If the variable is not found in the list with variables, or if the variable contains no text value, a non-zero value is returned.

GetVersion

**Umra.GetVersion**

**long GetVersion(**

**[out] long \* VersionMajor,**

**[out] long \* VersionMinor,**

**[out] long \* BuildNumber)**

Argument	Type	Description
VersionMajor	Out	The major version number of the UMRA COM object
VersionMinor	Out	The minor version number of the UMRA COM object.
BuildNumber	Out	The build number of the UMRA COM object.

The interface method retrieves version information of the UMRA COM object. The information must correspond with an eventually connected



**UMRA Service.** Otherwise, the COM object cannot connect to the **UMRA Service**.

HideVariable

**Umra.HideVariable**

**long HideVariable(BSTR VariableName);**

Argument	Type	Description
VariableName	In	The name of the variable for which the data must be hidden. Example: %Password%.

The interface method hides the data of the specified variable from being logged in log files.

On success, the return value is zero. If the specified variable does not exist or contains no data, a non zero value is returned.

LoadFormProject

**Umra.LoadFormProject**

**long LoadFormProject([in] BSTR FormProjectName, [in] IUnknown \* pFormProject)**

Argument	Type	Description
FormProjectName	In	The name of the project for which the form must be obtained.

pFormProject	in/out	The returned form project. The argument must be specified as another UMRA COM object: UmraFormProject. This COM object can then be used to obtain form table objects.
--------------	--------	---

The interface method retrieves the form of an **UMRA Form** project. The method is used to obtain table information from a table of an **UMRA Form** project.

On success, the return value is zero. If the specified form does not exist, or if the user has no access rights for the form, a nonzero value is returned.

ReleaseConnection

**Umra.ReleaseConnection**

**long ReleaseConnection()**

The interface method release the connection with the UMRA Service and terminates the *UMRA session* on page 81 that is maintained at the UMRA Service for this instance of the COM object.

RestoreConnection

**Umra.RestoreConnection**

**long RestoreConnection([in] BSTR ConnectionString)**

Argument	Type	Description
ConnectionString	Out	The text value retrieved with COM object method Umra.GetConnectionString

The interface method reconnects to the UMRA Service and restores the

*UMRA session* on page 81 that was initialized earlier. The input value should equal the value retrieved with COM method `Umra.GetConnectionString`.

`SetVariableBool`

**Umra.SetVariableBool**

**void SetVariableBool([in] BSTR VariableName, [in] VARIANT\_BOOL ValueBool)**

Argument	Type	Description
VariableName	in	The name of the variable set by this method. Example: %SomeFlag%.
ValueBool	in	The boolean value of the variable to set. Example: 0. A value of 0 corresponds with <b>false</b> . All other values correspond with <b>true</b> .

The interface method adds a boolean variable-value pair to the list of variables maintained by the COM object. The list is used to execute UMRA projects on the **UMRA Service**. To reset the list with variables, call method **ClearVariables**. The value of the variable must be either true or **false**. Other methods are available for different variable value data types.

When a variable with the same name already exists in the list with variables, it is overwritten by this method.

`SetVariableLong`

**Umra.SetVariableLong**

**void SetVariableLong([in] BSTR VariableName, [in] long ValueLong)**

Argument	Type	Description
VariableName	in	The name of the variable set by this method. Example: %SomeValue%.
ValueLong	in	The numeric value of the variable to set. Example: 5.

The interface method adds a numeric variable-value pair to the list of variables maintained by the COM object. The list is used to execute UMRA projects on the UMRA Service. To reset the list with variables, call method **ClearVariables**. The value of the variable must be numeric. Other methods are available for different variable value data types.

When a variable with the same name already exists in the list with variables, it is overwritten by this method.

SetVariableTable

**Umra.SetVariableTable**

**void SetVariableTable(BSTR VariableName, IUnknown\* ValueTable)**

Argument	Type	Description
VariableName	In	The name of the variable set by this method. Example: %UserTable%.
ValueTable	In	The data table to be stored in the variable list. The argument must be specified as one of the other UMRA COM object types: UmraDataTable. This COM object is used to copy the table data into the variable list.

The interface method adds a table variable-value pair to the list of variables maintained by the COM object. The list is used to execute UMRA projects on the **UMRA Service**. To reset the list with variables, call method **ClearVariables**. The value of the variable must be of the UMRA COM object type UmraDataTable. When a variable with the same name already exists in the list with variables, it is overwritten by this method. The following Visual Basic fragment shows how to use this method.

```
Dim Umra
...
Umra = CreateObject("UmraCom.Umra")
...
Dim UmraTable As UMRacomLib.UmraDataTable
UmraTable =
CreateObject("UmraCom.UmraDataTable")
...
[table manipulation statements]
...
Umra.SetVariableTable("%NewComTable%",
UmraTable)
```

SetVariableText

#### **Umra.SetVariableText**

**void SetVariableText([in] BSTR VariableName, [in] BSTR ValueText)**

Argument	Type	Description
VariableName	In	The name of the variable set by this method. Example: %FirstName%.
ValueText	In	The textual value of the variable to set. Example: John.

The interface method adds a textual variable-value pair to the list of variables maintained by the COM object. The list is used to execute UMRA projects on the **UMRA Service**. To reset the list with variables, call method **ClearVariables**. The value of the variable must be text. Other methods are available for different variable value data types.

When a variable with the same name already exists in the list with variables, it is overwritten by this method.

UmraCheckLicense

**Umra.UmraCheckLicense**

**C++:** public virtual HRESULT UmraCheckLicense(BSTR DomainOu, LONG DomainOuType, VARIANT\_BOOL \* LicenseValidFlag, LONG \* RetVal)

**VB:** Function UmraCheckLicense(ByVal DomainOu As String, ByVal DomainOuType As Integer, ByRef LicenseValidFlag As Boolean) As Integer

Argument	Type	Description
DomainOu	in	The name of the domain and organizational unit for which the license code is tested. The syntax depends on argument DomainOuType. For DomainOuType=0, the NETBIOS name must be specified, e.g. DOMAIN. For DomainOuType=1, the DNS/OU notation is used: domain.com, domain.com/ou/OtherOu
DomainOuType	in	Specification of the format of argument DomainOU. Possible values: 0: NETBIOS (DOMAIN, TOOLS4EVER) 1: DNS or DNS/OU (domain.com, domain.com/ou)
LicenseValidFlag	out	TRUE: A valid license is configured for the specified DomainOU argument. FALSE: No valid license is configured for the specified DomainOU argument.

The interface method is used to check if a valid UMRA license is configured for the specified domain/ou. Note that the UMRA COM object must be connected to a UMRA Service to successfully execute this method.

*UmraFormProject***UMRA COM object: UmraFormProject**

This UMRA COM object is used to access a table that is part of a form of an UMRA Form project. The COM object should not be used to access the data of a variable that holds table data. The UMRA COM object *UmraDataTable* must be used for table data variables instead.

*GetFormTable***UmraFormProject.GetFormTable**

**long GetFormTable([in] BSTR \* FormTableName, [in] IUnknown \* pFormTable)**

Argument	Type	Description
FormTableName	In	The name of the form table as defined in the UMRA form.
pFormTable	In/out	The retrieved form table. The argument must be specified as one of the other UMRA COM object types: <b>UmraFormTable</b> . This COM object can then be used to access the data of the table.

The interface method retrieves the table of a form. When the form of an UMRA Form project contains a table, this method is used to access the table of the form. To access the form table, a special UMRA COM object is used: **UmraFormTable**. An instance of such a COM object must be passed as one of the arguments of this method. Before this method is called, the method **Umra.LoadFormProject** must be executed first. On success, the form table data is copied into the passed COM object. The interface methods of the **UmraFormTable** object can then be used to access the table data.

On success, the method returns zero. If the form table is not found in the UMRA project maintained by the UMRA COM object, a non-zero value is returned.

*UmraFormTable***UMRA COM object: UmraFormTable**

This UMRA COM object is used to access the data of a form project table. See also **UmraFormProject.GetFormTable**.

The COM object should not be used to access the data of a variable holding table data. Instead, the UMRA COM object **UmraDataTable** should be used for table data variables.

*GetCellText***UmraFormTable.GetCellText**

**long GetCellText([in] long RowIndex, [in] long ColumnIndex, [out] VARIANT\* CellText)**

Argument	Type	Description
<b>RowIndex</b>	In	The index of the row (0,1,2,...,N-1) that contains the requested cell.
ColumnIndex	In	The index of the column (0,1,2,...,M-1) that contains the requested cell.
CellText	Out	The result text value of the requested cell.

The interface method retrieves the text value from one specific cell of the form table. The method returns zero on success. If the cell does not exist or when the cell value cannot be converted into text, a nonzero value is returned.

*UmraDataTable***UMRA COM object: UmraDataTable**

This UMRA COM object is used to access the table data of a variable. If the script of an UMRA Form is executed by UMRA COM (**Umra.ExecuteProjectScript**), a variable with table data can be



generated. This variable and table data value is returned to the UMRA COM project. To access the variable table data value, first the method **Umra.GetVariableDataTable** must be called. With this step, an instance of COM object **UmraDataTable** is initialized. Next, the single interface function of this COM object is used to access the table data.

#### AppendColumn

UmraDataTable.AppendColumn

**public virtual HRESULT AppendColumn( LONG \* RetVal)**

**Function AppendColumn() As Integer**

The interface method appends a column to the existing table. The method returns zero on success.

#### AddRow

UmraDataTable.AddRow

**public virtual HRESULT AddRow( LONG \* RetVal)**

**Function AddRow() As Integer**

The interface method adds an empty row at the end of the table. The method returns zero on success.

#### CreateTable

UmraDataTable.CreateTable

**public virtual HRESULT CreateTable( LONG Rows, LONG Columns, LONG \* RetVal)**

**Function CreateTable(ByVal Rows As Integer, ByVal Columns As Integer) As Integer**

Argument	Type	Description
Rows	In	The number of rows of the table to create.
Columns	In	The number of columns of the table to create.

The interface method creates a table with the specified number of columns and rows. Initially, all cells of the table are filled with empty string values. The method returns zero on success.

CreateTable2

UmraDataTable.CreateTable2

**public virtual HRESULT CreateTable2( VARIANT ColumnNames, LONG \*RetVal)**

**Function CreateTable2(ByVal ColumnNames As Object) As Integer**

Argument	Type	Description
ColumnNames	In	The names of the columns of the new table. The number of entries determines in the array equals the number of columns of the new table.

The interface method creates a table. The ColumnNames input argument specifies the names of the columns. Initially, the table contains no rows. The method returns zero on success. The following fragment shows how to use this method in Visual Basic:

```
....
Dim ColumnNames(4) As String
ColumnNames(0) = "SAM"
ColumnNames(1) = "Phone"
ColumnNames(2) = "Address"
ColumnNames(3) = "City"
```

```
UmraTable.CreateTable2(ColumnNames)
```

....

The method `Umra.SetVariableTable` can be used to store the table in the variable list.

`CreateTable3`

`UmraDataTable.CreateTable3`

**public virtual HRESULT CreateTable3( LONG ColumnCount, LONG \*RetVal)**

**Function CreateTable3(ByVal ColumnCount As Integer) As Integer**

Argument	Type	Description
ColumnCount	In	The number of columns of the table to create..

The interface method creates a table with the specified number of columns. Initially, the table contains no rows. The method returns zero on success.

`GetCellText`

`UmraDataTable.GetCellText`

**C++: public virtual HRESULT GetCellText( LONG RowIndex, LONG ColumnIndex, VARIANT \* CellText, LONG \* RetVal)**

**VB: Function GetCellText(ByVal RowIndex As Integer, ByVal ColumnIndex As Integer, ByRef CellText As Object) As Integer**

Argument	Type	Description
RowIndex	In	The index of the row (0,1,2,...,N-1) that contains the requested cell.

ColumnIndex	In	The index of the column (0,1,2,...,M-1) that contains the requested cell.
CellText	Out	The result text value of the requested cell.

The interface method retrieves the text value from one specific cell of the table data. The method returns zero on success. If the cell does not exist or when the cell value cannot be converted into text, a nonzero value is returned.

GetCellTextEx

UmraDataTable.GetCellTextEx

**C++:** `public virtual HRESULT GetCellTextEx( LONG RowIndex, BSTR ColumnName, VARIANT * CellText, LONG * RetVal)`

**VB:** `Function GetCellTextEx(ByVal RowIndex As Integer, ByVal ColumnName As String, ByRef CellText As Object) As Integer`

Argument	Type	Description
RowIndex	In	The index of the row (0,1,2,...,N-1) that contains the requested cell.
ColumnName	In	The name of the column that contains the requested cell.
CellText	Out	The result text value of the requested cell.

The interface method retrieves the text value from one specific cell of the table data. The cell is specified by the row index and column name. The method returns zero on success. If the cell does not exist or when the cell value cannot be converted into text, a nonzero value is returned.

### GetColumnCount

UmraDataTable.GetColumnCount

**public virtual HRESULT GetColumnCount( LONG \* ColumnCount!)**

**Function GetColumnCount() As Integer**

Argument	Type	Description
ColumnCount	Out	The number of columns of the table.

The interface method retrieves the number of columns of the table. If an error occurs, -1 is returned.

### GetColumnName

UmraDataTable.GetColumnName

**public virtual HRESULT GetColumnName( LONG ColumnIndex,  
VARIANT \* ColumnName, LONG \* RetVal)**

**Function GetColumnName(ByVal ColumnIndex As Integer, ByRef  
ColumnName As Object) As Integer**

Argument	Type	Description
ColumnIndex	In	The index of the column (0,1,2,...,M-1) of which the name must be retrieved.

The interface method retrieves the name of the specified column. The column is specified by its index. The method returns zero on success.

GetRowCount

UmraDataTable.GetRowCount

**public virtual HRESULT GetRowCount( LONG \* RowCount)**

**Function GetRowCount() As Integer**

Argument	Type	Description
RowCount	Out	The number of rows of the table.

The interface method retrieves the number of rows of the table. If an error occurs, -1 is returned.

SetCellText

UmraDataTable.SetCellText

**public virtual HRESULT SetCellText( LONG RowIndex, LONG ColumnIndex, BSTR CellText, LONG \* RetVal)**

**Function SetCellText(ByVal RowIndex As Integer, ByVal ColumnIndex As Integer, ByVal CellText As String) As Integer**

Argument	Type	Description
RowIndex	In	The index of the row (0,1,2,...,N-1) that contains the requested cell.
ColumnIndex	In	The index of the column (0,1,2,...,M-1) that contains the requested cell.
CellText	Out	The result text value of the requested cell.

The interface method retrieves the text value from one specific cell of the table data. The method returns zero on success. If the cell does not exist or when the cell value cannot be converted into text, a nonzero value is returned.

## SetColumnName

UmraDataTable.SetColumnName

**public virtual HRESULT SetColumnName( LONG ColumnIndex, BSTR ColumnName, LONG \* RetVal)**

**Function SetColumnName(ByVal ColumnIndex As Integer, ByVal ColumnName As String) As Integer**

Argument	Type	Description
ColumnIndex	In	The index of the column (0,1,2,...,M-1) for which the name is set.
ColumnName	In	The name of the column

The interface method sets the name of an existing column. The method returns zero on success.

## SetColumnNameEx

UmraDataTable.SetColumnNameEx

**public virtual HRESULT SetColumnNameEx( BSTR CurrentName, BSTR NewName, LONG \* RetVal)**

**Function SetColumnNameEx(ByVal CurrentName As String, ByVal NewName As String) As Integer**

Argument	Type	Description
CurrentName	In	The index of the row (0,1,2,...,N-1) that contains the requested cell.
NewName	In	The index of the column (0,1,2,...,M-1) that contains the requested cell.

The interface method replaces the name of an existing column with a new name. The method returns zero on success.



---

### 3.3.3. UMRA COM in VB scripts

This section explains how to use UMRA COM in Visual Basic scripts.

#### Introduction

A very common usage of UMRA COM is in Visual Basic scripts. Visual Basic is a very general development environment to create Windows applications and scripts. Managing Active Directory directly from Visual Basic programs and scripts is possible but rather complex and not secure.

The integration of Visual Basic and User Management Resource Administrators using UMRA COM offers a very flexible and powerful solution to manage the complex Active Directory with simple and robust program. The complex Active Directory tasks are implemented with UMRA projects and executed by the **UMRA Service**. The projects are initiated using UMRA COM from within Visual Basic scripts/programs.

Since the UMRA projects are executed by the **UMRA Service** in such an environment, the Visual Basic scripts and programs can be delegated to less experienced users. The **UMRA Service** will check the access rights of the connecting user account before the UMRA project script is executed.

The technique to use **UMRA COM** objects in Visual Basic scripts is best described with an example.

#### Example project - Goal

Goal of the example application is to create a number of user accounts in Active Directory for which the first and last name are available from a table in an MS-Access database.

In this example, a simple UMRA project is configured on the **UMRA Service**. The project creates a user account in Active Directory. The input of the project is the first and last name of the new user account. The input values are passed using the variables **%FirstName%** and **%LastName%**.

The MS-Access database contains a table with first and last names. Goal is to create the user accounts for all table entries. A Visual Basic script with **UMRA COM** objects is used to read the data from the MS-Access database and create a user account for each row.

### Configuring the UMRA project

In the example, an UMRA project is used to create a user account. The project takes two input variables: %FirstName% and %LastName%. From these names, the project generates a unique user account name (%UserName%) and creates the user account. Also, a random password (%Password%) is generated.

The UMRA project is a form project that contains only a script, not a form. The UMRA form project is contained in file

.\Example Projects\Automation\VBScript\MsAccess\ CreateAccount.ufp relative to the **UMRA Console** program directory.

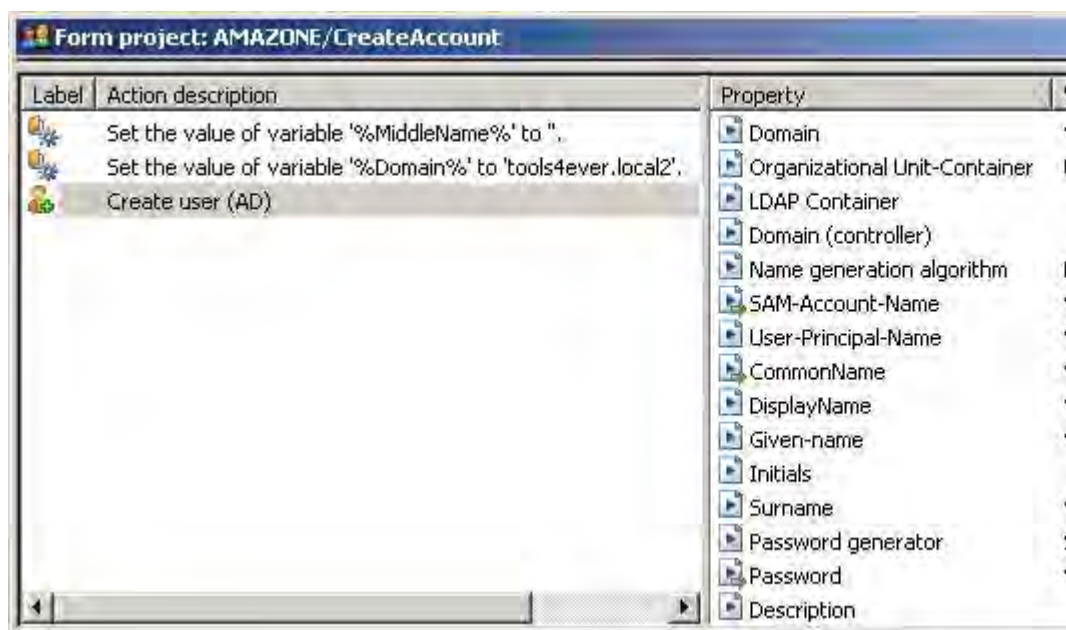


Figure 2: The script of the UMRA Form project used to create the account.

The script contains 3 lines only. In the first 2 lines the following variables are set: %MiddleName% and %Domain%. The %MiddleName% variable is set to an empty string. The %MiddleName% is used in the name generation algorithm and always specified as an empty string. The %Domain% is set equal to the DNS name of the domain in which the user accounts are created: **tools4ever.local2**. You need to adjust this value to make the project work in your own environment.

### Create user action

The **Create user (AD)** script action is the main action of the form project. The action creates the user account in the specified domain. The inputs of the script (action) are the values of the variables **%FirstName%** and **%LastName%**. These variables are used by the name generation algorithm. The **Default** name generation algorithm is specified for this property.

The algorithm uses 3 input variables: **%FirstName%**, **%MiddleName%** and **%LastName%**. The **%MiddleName%** is always empty, the **%FirstName%** and **%LastName%** variables are specified in the Visual Basic Script. The outputs of the name generation algorithm are the values of the variables **%UserName%** and **%FullName%**. These values are used to create the user account. The **%UserName%** variable is exported to the list of variables. The password of the new user account is generated as part of the action and exported to variable **%Password%**.

In UMRA, variables play an important role. Some variables are used as input and others as output variables. Other variables are generated by the script and only used in the script actions. The interface functions of the UMRA COM objects support input and output variables.

### Input and output variables

The following table lists the input and output variables that are used by this example project and communicated with UMRA COM.

Variable	Type	Description
%FirstName%	Input	First name of the user account that must be created. Specified in the database read by the Visual Basic script.
%LastName%	Input	Last name of the user account that must be created. Specified in the database read by the Visual Basic script.

%UserName%	Output	The resulting name (SAM account name) of the created user account. Presented to the end-user in the browser.
%Password%	Output	The password generated and set for the created user account. Presented to the end-user in the browser.

Table 2: In- and output variables of the example project.

Note that for the form project, security access rights must be configured: The end-users that are allowed to run the script of this project must be configured. The end-users are the user accounts logged on to the computer that run the internet explorer and access the UMRA application.

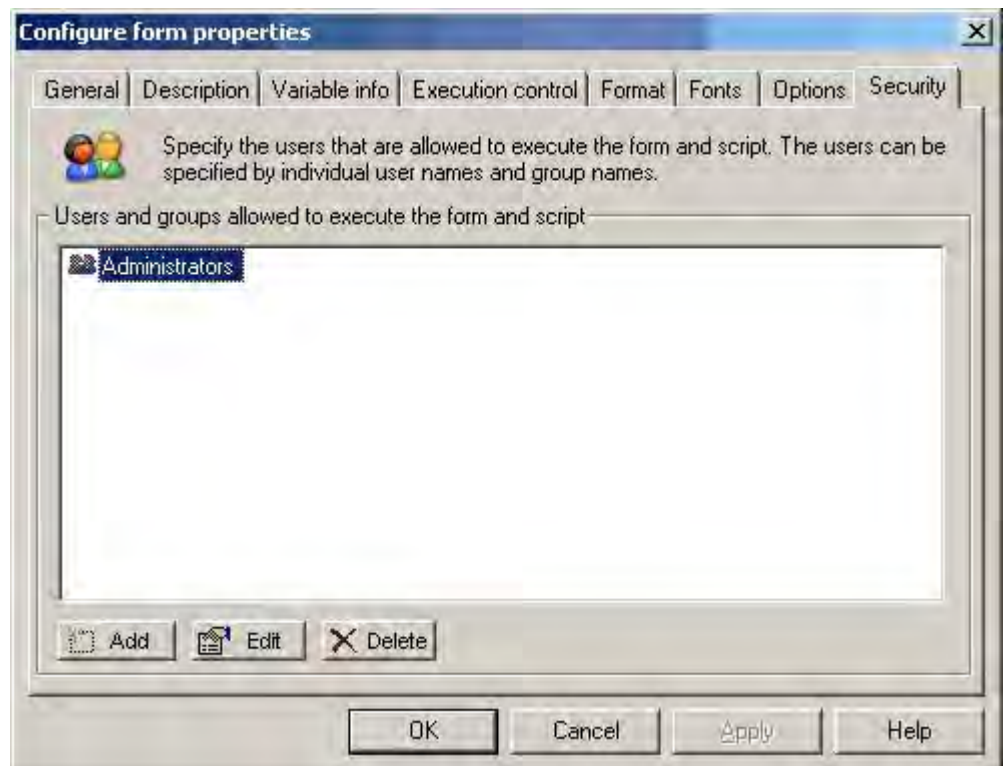


Figure 3: The security access rights specified for the UMRA project that creates the account.

In this example project, the **Administrators** are configured as the users that are allowed to run this project.

## MS Access database

### Jet engine

The database is a simple MS-Access (Jet-engine) database with a single table only. The database can be found in the following location:

.\\Example Projects\\Automation\\VBScript\\MsAccess\\FirstLastNames.mdb  
relative to the UMRA Console program directory. The database holds table **FirstLastNamesTable** with the columns **Id**, **FirstName**, and **LastName**.



Id	FirstName	LastName
1	John	Smith
2	Jean	Williams
3	George	Hush
4	Dennis	Taylor
5	Ella	Parker
6	Jane	Henderson
7	John	Doe
(AutoNumber)		

Record: 8 of 8  
Datasheet View

### Microsoft Office Access

Note that in order to use the database with UMRA in Visual Basic Script, you do not need to have Microsoft Office Access installed. The Jet-engine to access the database is installed by default.

### Visual Basic script

The Visual Basic script is a standard script that uses a database connection to read the MS-Access database. UMRA COM objects are used to execute the project that creates a user account on the **UMRA Service**.

The Visual Basic script can be found at the location

**.\\Example Projects\\Automation\\VBScript\\MsAccess\\  
CreateAccountAccess.vbs**

relative to the **UMRA Console** program directory.

The following listing shows the complete script. Note that the script is kept as simple as possible to make it easier to understand. More error handling should be added to the script in operational environments.

```
Dim adoConn, adoRS

Set adoConn = CreateObject("ADODB.Connection")

Set adoRS = CreateObject("ADODB.Recordset")

adoConn.Provider = "Microsoft.Jet.OLEDB.4.0"

adoConn.Open "FirstLastNames.mdb"

Set adoRS.ActiveConnection = adoConn

adoRS.Open "SELECT * FROM FirstLastNamesTable"


Dim Umra, Username, UserPassword

Set Umra = CreateObject("UmraCom.Umra")

RetVal = Umra.Connect("AMAZONE", 56814)


While adoRS.EOF = False

    Umra.SetVariableText "%Firstname%", adoRS("FirstName")

    Umra.SetVariableText "%LastName%", adoRS("LastName")

    RetVal=Umra.ExecuteProjectScript("CreateAccount")

    RetVal=Umra.GetVariableText("%Username%", Username)

    RetVal=Umra.GetVariableText("%Password%", UserPassword)

    wscript.echo "User created: " & Username & " - " & UserPassword

    adoRS.MoveNext
```

Wend

If RetVal = 0 Then

    WScript.Echo "Project executed successfully, code: " & RetVal

Else

    WScript.Echo "Project execution failed, code: " & RetVal

End If

In the next section, parts of the scripts are shown with some comments for each section.

#### **Script section: Setting up the database connection**

##### **ADO database connection – OLE DB provider**

The first section of the script is used to setup a connection with the database. In this script, the Microsoft ADO (ActiveX Data Objects) standard is used to access the database. This is a very common technique used in VBScript to access a database. With ADO, the data is accessed using the OLE DB provider. Almost all database types can be accessed using OLE DB (including ODBC connections) so this method can be used for almost all databases.

```
Dim adoConn, adoRS
```

```
Set adoConn = CreateObject("ADODB.Connection")
```

```
Set adoRS = CreateObject("ADODB.Recordset")
```

```
adoConn.Provider = "Microsoft.Jet.OLEDB.4.0"
```

```
adoConn.Open "FirstLastNames.mdb"
```

```
Set adoRS.ActiveConnection = adoConn
```

```
adoRS.Open "SELECT * FROM FirstLastNamesTable"
```

Two script variables are initialized: the connection (adoConn) and the record set (adoRS). With the **CreateObject** function, the ADODB COM objects **ADODB.Connection** and **ADODB.RecordSet** are created. The

**CreateObject** function creates an instance of the specified COM object. So this is also COM, not UMRA COM but ADO COM.

Next, the **Provider** of the connection object is set to **Microsoft.Jet.OLEDB.4.0**. The provider member specifies the type of the database connection. For a different database, this provider specification differs. The database is opened with the **Open** method of the **ADODB.Connection** object. The name of the database file, **FirstLastNames.mdb** is specified as the argument of the call.

With the statement **Set adoRS.ActiveConnection = adoConn** the recordset object is initialized. The **Open** method of the **ADODB.Recordset** is then used to perform a query in the database: **SELECT \* FROM FirstLastNamesTable**: All records from the table are returned and can be accessed through the recordset object.

#### **Script section: Connecting to the UMRA Service**

The database connection is not initialized. For each record returned from the database, a user account must be created. Before the loop to create the accounts is implemented, a connection must be setup with the **UMRA Service**. This is done by creating the **Umra** COM object of UMRA COM and calling the **Connect** method of the object.

```
Dim Umra, Username, UserPassword
```

```
Set Umra = CreateObject("UmraCom.Umra")
```

```
RetVal = Umra.Connect("AMAZONE", 56814)
```

First, the variables are declared. The **Umra** variable will hold the COM object. The variables **Username** and **UserPassword** will hold the text values of the user name and password for the created user account. The values will be shown to the end user

The UMRA COM object **Umra** is then created with the **CreateObject** call. The argument **UmraCom.Umra** specifies the UMRA COM library and the type of object (**Umra**) of which an instance must be created. If the UMRA COM library is not installed and/or registered, the **CreateObject** call will fail.

The UMRA COM object now connects to the **UMRA Service** with the statement **RetVal=Umra.Connect("AMAZONE",56814)**. The **Connect** method takes two arguments: the name of the computer that runs the



**UMRA Service** and the port number used by the **UMRA Service**. 56814 is the default **UMRA Service** port number.

On success, the return value of the **Connect** method is zero. The UMRA COM object is now connected to the **UMRA Service**.

**Script section: Executing projects on UMRA Service**

The database connection is initialized and the UMRA COM object is connected to the **UMRA Service**. Now the loop is implemented. For each returned database record, the variables list of the UMRA COM object is initialized and the **UMRA Service** is requested to execute the UMRA project that creates the user account.

While adoRS.EOF = False

```
    Umra.SetVariableText "%Firstname%", adoRS("FirstName")

    Umra.SetVariableText "%LastName%", adoRS("LastName")

    RetVal=Umra.ExecuteProjectScript("CreateAccount")

    RetVal=Umra.GetVariableText("%Username%", Username)

    RetVal=Umra.GetVariableText("%Password%", UserPassword)

    wscript.echo "User created: " & Username & " - " & UserPassword

    adoRS.MoveNext
```

Wend

The **While ... Wend** construction is used for the loop. The loop is terminated when no more records are found: when **adoRS.EOF=False** no longer holds.

For each cycle of the loop, first the list with variables maintained by the UMRA COM object is initialized. The method **SetVariableText** is used. The method takes two arguments: the name of the variable (**%FirstName%**) and the value. The value is copied from the record set: **adoRS("FirstName")**. Here, **FirstName** is the name of the column of the database table from which the value must be obtained. Using this method, the **%FirstName%** and **%LastName%** UMRA variables are initialized.

With method **Umra.ExecuteProjectScript** the **UMRA Service** is requested to execute the passed project. With some other information, the list

with variables and values is sent to the **UMRA Service**. The **UMRA Service** will check the access rights of the requesting user account, load the project and execute the script of the project. The variables updated or generated by the script of the project are returned to the UMRA COM object. On success, the return value is zero.

The **Umra.GetVariableText** method of the UMRA COM object is now used to retrieve the values of the variables **%UserName%** and **%Password%**. The values are stored in the script variables **Username** and **UserPassword**. With the standard **wscript.echo** method, the user name and password of the created user account is presented to the end user.



For each user account created, the message above is shown. In a more practical script, this is probably not convenient and should be changed.

Finally, the record set moves to the next record: **adoRS.MoveNext**.

#### **Testing and executing the script**

To execute the script, log on to a computer of the domain with an administrative Active Directory account. Update or enter the Visual Basic script with your favorite editor. Make sure the **UMRA Service** maintains the project referenced in the script and that the UMRA COM object connects to the appropriate UMRA Service. To execute the script, open a command prompt and enter the name of the script:

**CreateAccountAccess.vbs.**

This will automatically initialize **Windows Scripting Host** to execute the script. For each account created, the **UMRA Service** is requested to execute the UMRA project and create the account. A message box is shown for every account that is created.

The **UMRA Service** log file will contain the log information generated by the service when the script is executed. A listing for a single user account is shown below.

14:57:14 01/05/2006 Variable 1: %Firstname%=John

14:57:14 01/05/2006 Variable 2: %LastName%=Smith

14:57:14 01/05/2006 Variable 3:

%UmraFormSubmitAccount%=T4ELOC2\Administrator

14:57:14 01/05/2006 Creating AD account in specified domain:  
'tools4ever.local2'.

14:57:15 01/05/2006 Creating AD account in container 'Users'.

14:57:15 01/05/2006 Creating AD account in Organizational Unit-Container:  
'LDAP://CN=Users,DC=tools4ever,DC=local2'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2': Using name generation algorithm  
'Default', 100 iterations maximum for duplicate names.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Common name of user set to  
'John Smith'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. SAM account name (username) of  
user set to 'smithj'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute  
'userPrincipalName' of object 'John Smith' set to 'smithj@tools4ever.local2'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute 'displayName' of  
object 'John Smith' set to 'John Smith'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute 'givenName' of  
object 'John Smith' set to 'John'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute 'sn' of object 'John  
Smith' set to 'Smith'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Password generated ('Strong, 7 chars with 1 numeric, 1 special (variable: %Password%)') and set in variable '%Password%'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Account disabled'=FALSE (101034). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Password never expires'=FALSE (101032). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Store password using reversible encryption'=FALSE (101033). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Smart card is required for interactive logon'=FALSE (101035). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Account is trusted for delegation'=FALSE (101036). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Account is sensitive and cannot be delegated'=FALSE (101037). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Use DES encryption types for this account'=FALSE (101038). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Don't require Kerberos preauthentication'=FALSE (101039). Result not changed.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Account expiration date not specified for new user 'John Smith' object.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. User 'John Smith' successfully created.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Password set for new user object  
'John Smith'.

14:57:15 01/05/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Flag 'User Cannot Change  
Password' not changed from default value (FALSE).

14:57:15 01/05/2006 Form message:  
'01/05/2006,14:57:14,T4ELOC2\Administrator,"Automation run  
script",OK,CreateAccount'



---

#### 3.3.4. UMRA COM in IIS

This section explains how to use UMRA COM together with the Microsoft Internet Information Server (IIS).

##### Introduction

The functions of User Management Resource Administrator can be accessed through an internet browser by using an **UMRA COM** object in IIS ASP pages.

In such an environment: 3 components are involved:

1. The internet browser that shows the web-pages and is used to enter input fields;
2. The IIS web-server to which the browser connects;
3. The **UMRA Service**, contacted by the **UMRA COM** objects contained on the ASP pages running on the IIS web-server.

As an example, consider a computer running **Internet Information Services (IIS)**. One of the websites maintained by IIS is used to create a user account with UMRA. To implement such an application, the website contains ASP pages with **UMRA COM** objects to access the **UMRA Service**.



Figure 4: Example project to create a user account with UMRA using a browser.

When the end user enters the first, middle and last name and clicks the **Create user** button, the browser information is sent to Internet Information Services. The ASP page that processes the input from the browser creates an UMRA COM object. Through the interface functions

of the COM object, the **UMRA Service** is contacted. The variables and values are initialized and the project is executed by the **UMRA Service**.

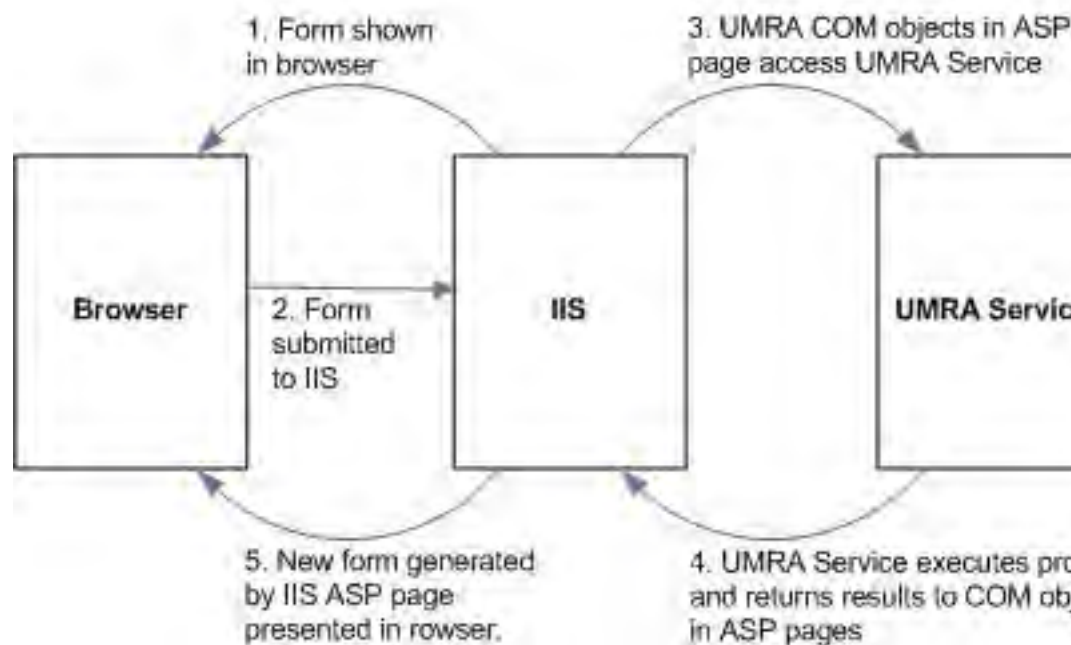


Figure 5: Sequence of steps of an UMRA application with UMRA COM on IIS. The ASP pages of the IIS website contain UMRA COM objects that communicate with the UMRA Service.

In this chapter, the example shown is described in great detail for Windows 2003/2000. All files of the example project can be found at the following location:

.\Example Projects\Automation\ASP\CreateAccount

relative to the **UMRA Console** program directory.

### Security and authentication

When running websites with ASP pages containing UMRA COM objects, the preferred authentication method is **integrated Windows authentication**. With such a configuration, the most simple and secure configuration is established.

The internet browser used is the Microsoft Windows Internet Explorer. On the computer that runs the Internet Explorer, a user is logged on to Active Directory. When the browser connects to IIS, the scripts contained by the ASP pages are executed on behalf of this user account.



So when the **UMRA COM** object is used to access the **UMRA Service**, the **UMRA Service** will check the access rights for this target user account.

### IIS configuration Windows 2003

**Note:** UMRA COM is available for both 32- and 64-bit platforms. See *UMRA COM on 64-bit platforms* on page 73 for more information.

This topic describes how to setup IIS on a computer running Window Server 2003, Standard Edition. The procedure is similar for computer running other versions of Windows, including all versions of Windows 2000.

1. When IIS is not already installed, log on as an administrator. Select **Start, Control Panel, Add or Remove Programs**.
2. Click the button **Add/Remove Windows Components**. In the **Windows Components Wizard** window, enable and select **Application Server** from the list.



Figure 6: Setting up IIS by enabling Application Server.

3. Click **Next** and **Finish** to install the selected options. To complete the setup, you need the Windows installation CD. When done, the list with services running on the computer will list the IIS service: **World Wide Web Publishing Service**.

When ready, the IIS Service needs to be configured.

4. Select **Start, Administrative Tools, Internet Information Services (IIS) Manager**. Browse to the current computer and select **Web Service Extensions**.

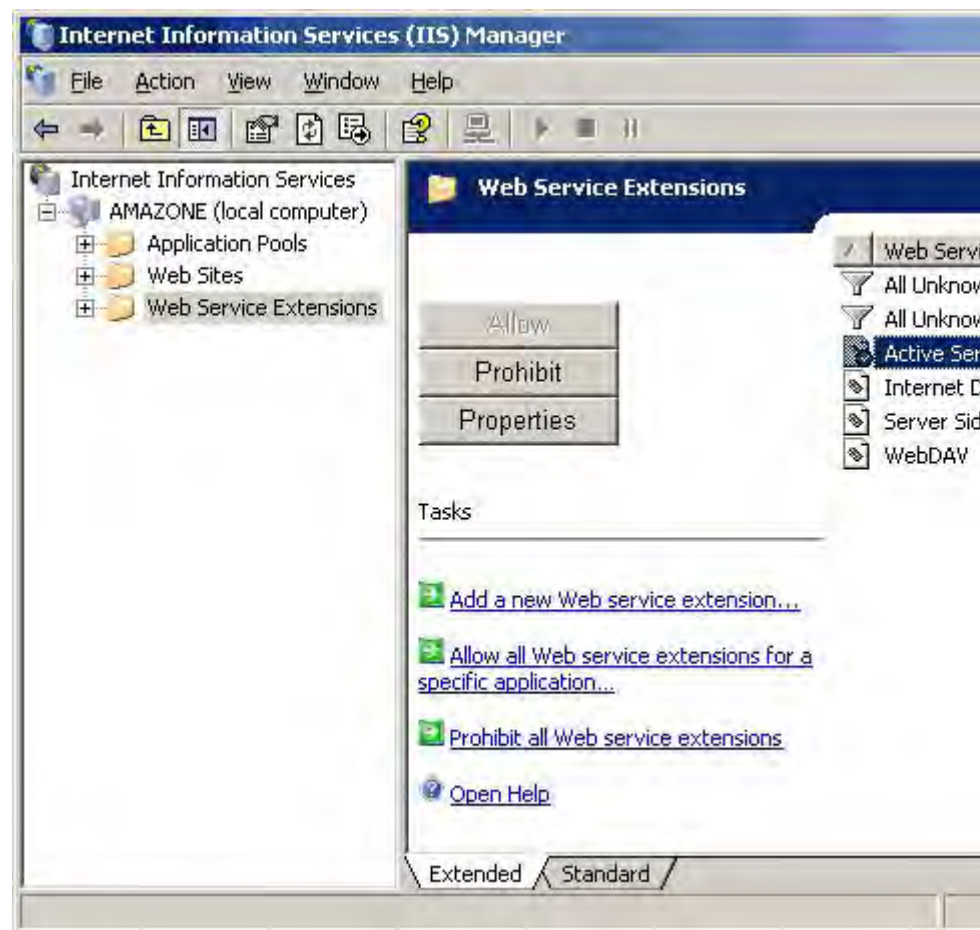


Figure 7 Allow Active Server Pages to run on IIS.

5. From the list with **Web Service Extensions**, select **Active Server Pages** and click the **Allow** button. This will allow ASP pages to run as part of a website maintained by the IIS server.

#### Creating the website

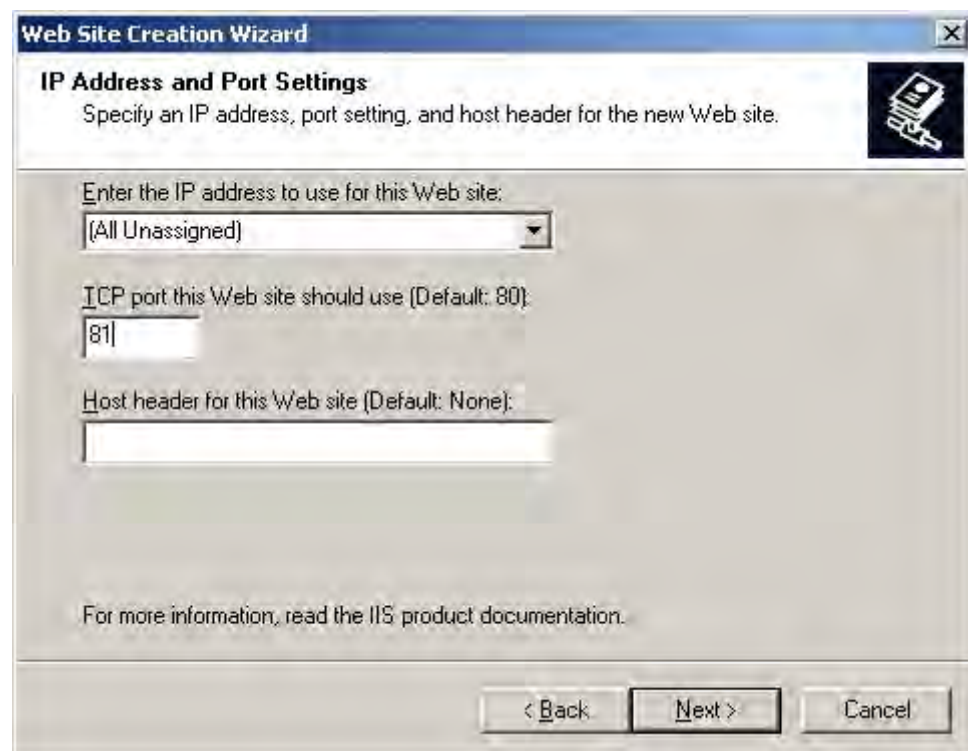
1. Create a folder where the UMRA website files will be stored. In this example this folder is: E:\UMRA\WebSite.

2. Start the **Internet Information Service Manager**, browse to the computer and right click option **Websites**. Select the menu option **New, Website....** Follow the wizard instructions to setup a basic website.



Figure 8: Enter any description of the website.

3. The description of the website is not that important. Just enter some text and click **Next** to continue.



The image shows a Windows-style dialog box titled "Web Site Creation Wizard". The main heading is "IP Address and Port Settings". Below the heading is a sub-instruction: "Specify an IP address, port setting, and host header for the new Web site." There are three input fields: 1. "Enter the IP address to use for this Web site:" with a dropdown menu currently showing "(All Unassigned)". 2. "TCP port this Web site should use (Default: 80):" with a text box containing "81". 3. "Host header for this Web site (Default: None):" with an empty text box. At the bottom, there is a line of text: "For more information, read the IIS product documentation." and three buttons: "< Back", "Next >", and "Cancel".

Figure 9: Specify the TCP port to be used by the website.

In the wizard window to enter the **IP Address and Port Settings** you do not need to change the default options, except for the TCP port the website should use. Each website running on the computer must have a unique port. The default website that is automatically installed uses default port 80. In order to use default port 80, you must first stop the default website to prevent port conflicts.

4. In the example shown, port 81 is entered. Click **Next** to continue with the **website home directory** specification.



Figure 10: Enter the home directory of the website. The directory must exist but does not have to contain any files yet.

5. Enter the name of the folder that is going to contain the website ASP and HTML files. In the example shown, this folder is E:\UMRA\WebSite. The folder needs to exist but does not need to contain any files yet. Click **Next** to continue with the configuration of the **Website Access Permissions**.

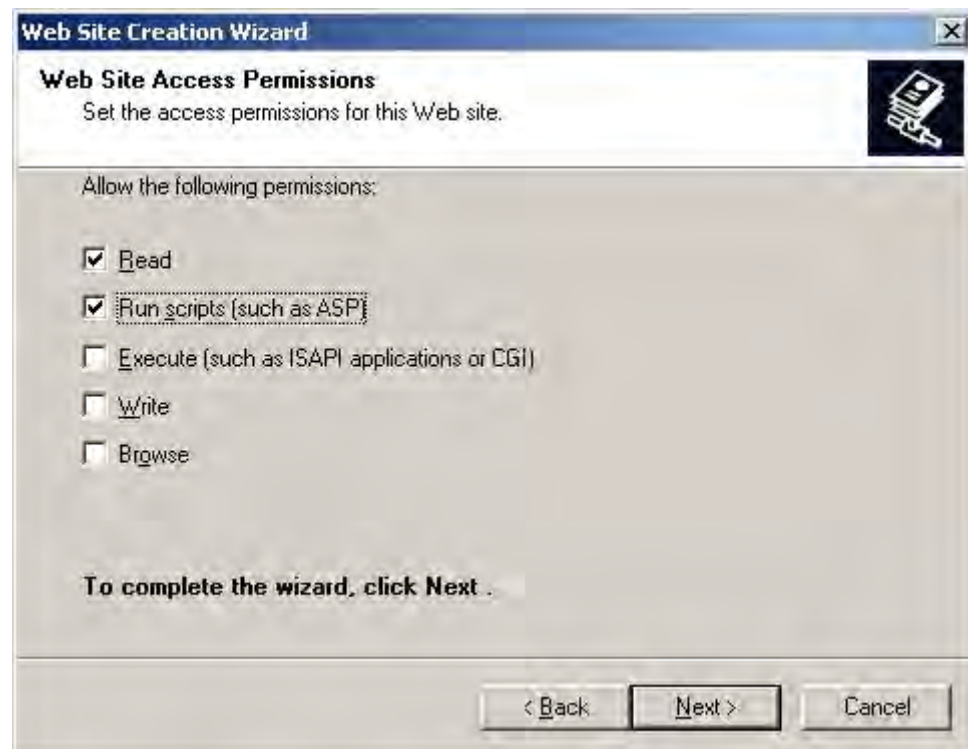


Figure 11: Specify the permissions of the new website.

6. You need to specify the **Read** and **Run scripts** permissions. The **Run scripts** permissions are required to allow ASP pages to run. You can specify additional permissions but this is not required. Click **Next** to complete the **Website Creation Wizard**. The website is now created as shown in the **Internet Information Services (IIS) Manager**.

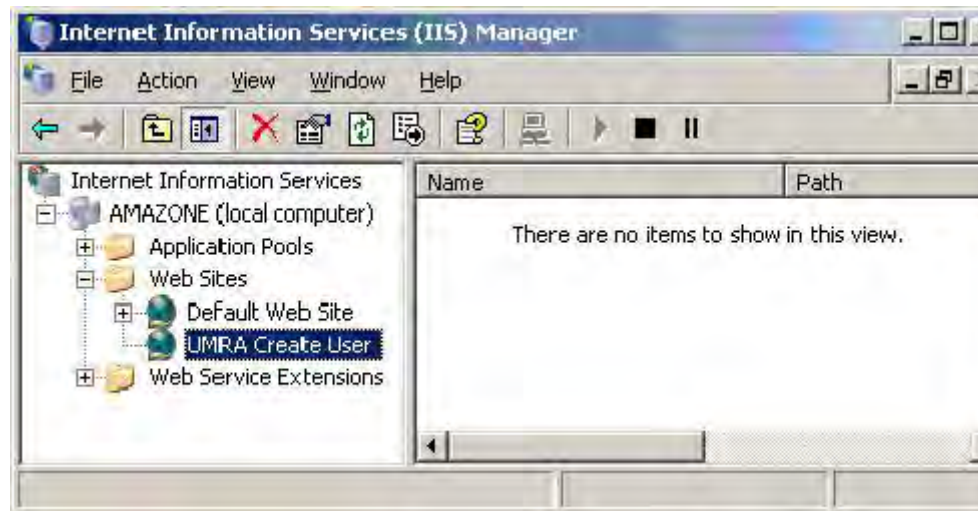


Figure 12: The new website is created and shown in the Internet Information Services Manager.



7. Finally, you need to configure the authentication method for the website. In order to be able to check the access rights, the ASP pages must be executed in the security context of the end-user of the browser. Right click the website and select **Properties**. Select **Directory security** and click the **Edit...** button in section **Authentication and access control**.

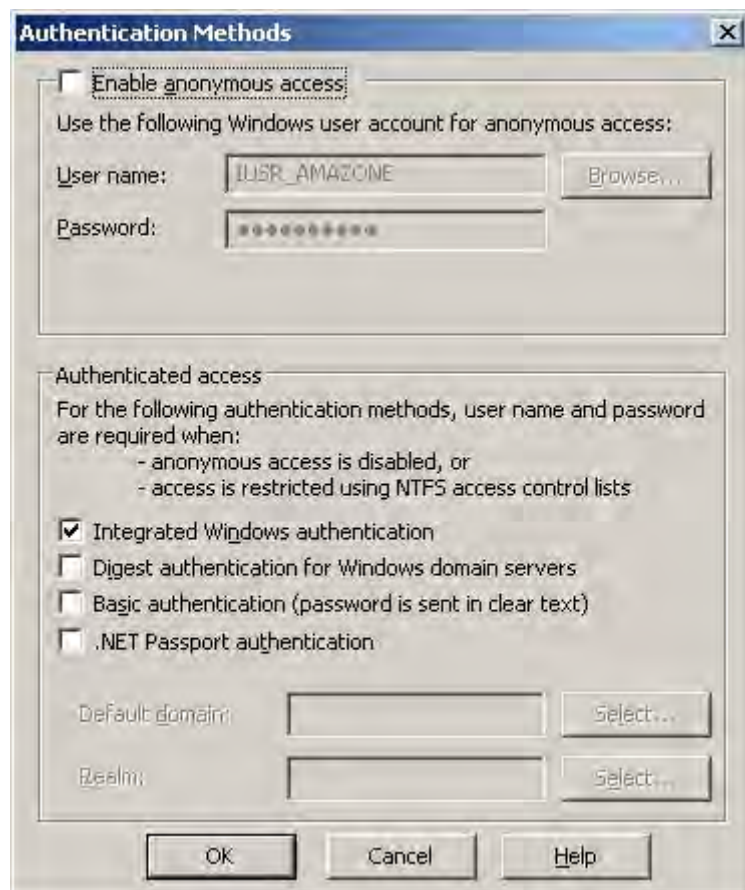


Figure 13: Specify the Authentication method: Disable anonymous access and enable integrated Windows authentication.

8. Disable the option **Enable anonymous access** and enable option **Integrated Windows authentication**. Click **OK** twice to exit the configuration.
9. Later, you can always change one of the configured options of the website: Right click the website and select **Properties**.

#### Configuring the UMRA project

The UMRA project is a form project that contains only a script, not a form. The UMRA form project is contained in file



**.\Example****Projects\Automation\ASP\CreateAccount\CreateAccount.ufp**

relative to the UMRA Console program directory. The UMRA project is the same as the one used for the example project used for the Visual Basic script example. See section *Configuring the UMRA project* on page 34 on page **Error! Bookmark not defined.** for more information.

**Setting up the IIS website**

The UMRA project is now ready and the (empty) website exists. You can now set up the website contents to complete the configuration.

For demonstration purposes, the website is kept very simple and contains only 2 pages:

**CreateAccount.asp:** The page that shows the input fields for first and last name and a button to submit the form. This is the first page the end-user connects to when accessing the website.

**ShowResults.asp:** The page that actually creates the account using UMRA COM and shows the results (%UserName% and %Password%) to the end-user in the browser. The page is generated as a response when the end-user clicks the submit button of the previous page.

Since the website pages contain ASP code, the pages have the .ASP extension. You can create the pages with your favourite editor or copy and edit the pages from the location

.\Example Projects\Automation\ASP\CreateAccount

relative to the **UMRA Console** program directory. The resulting ASP pages must be stored in the specified home directory of the new website.

Note that an ASP pages generates HTML code that is presented in a browser. Instead of the literal HTML text, the ASP page can contain script sections that produce HTML. The UMRA COM objects are used in the ASP script sections.

**Website page: CreateAccount.asp**

This the first page of the website. It does not contain UMRA COM objects but it contains the form and input fields that are needed to initialize the variables.

```
<HTML>

<HEAD>

<Title>Create User Account</TITLE>

</HEAD>


<FORM NAME=CreateAccount ACTION="ShowResults.asp" METHOD="POST">

First name: &nbsp;<INPUT TYPE="TEXT" NAME="FirstName">

<BR>

Last name: &nbsp;<INPUT TYPE="TEXT" NAME="LastName">

<BR>

<BR>

<INPUT TYPE="SUBMIT" VALUE="Create Account">

</FORM>
```

The page contains completely standard HTML. It contains a short header section (<HEAD>) and a form. The name of the form is CreateAccount and the action executed when the form is submitted is the generation and presentation of page ShowResults.asp.

The form contains two text input fields with names **FirstName** and **LastName**. The button is a submit type button with text **Create Account**.

This website page does not contain any specified ASP code. It is all straightforward HTML.

The page shows a title, some text, two input fields and a button. When the button is pressed, the content of the input fields is sent to the IIS server and the page ShowResults.asp is generated and presented.

**Website page: ShowResults.asp**

This page actually contains the UMRA COM objects and creates the user account. The contents of the page are listed below:

```
<HTML>
```

```
<HEAD>
```

```
<Title>User Account Created</TITLE>
```

```
</HEAD>
```

The user account created:

```
<%
```

```
    Set Umra = Server.CreateObject("UMRAcom.Umra")
```

```
    RetVal=Umra.Connect("AMAZONE",56814)
```

```
    Umra.SetVariableText "%FirstName%",Request.Form("FirstName")
```

```
    Umra.SetVariableText "%LastName%",Request.Form("LastName")
```

```
    RetVal=Umra.ExecuteProjectScript("CreateAccount")
```

```
    RetVal=Umra.GetVariableText("%UserName%",UserName)
```

```
    Response.Write "<BR>"
```

```
    Response.Write "User name: "
```

```
    Response.Write UserName
```

```
    RetVal=Umra.GetVariableText("%Password%",Password)
```

```
    Response.Write "<BR>"
```

```
    Response.Write "Password: "
```

Response.Write Password

%>

<FORM NAME=NextUser ACTION="CreateAccount.asp" METHOD="POST">

<INPUT TYPE="SUBMIT" VALUE="Next user">

</FORM>

</HTML>

The page contains 3 sections:

1. **the header section;**
2. **the ASP section**
3. **a form section.**

#### Explanation of the ASP code

The header section is straightforward and contains the title of the page only. The ASP section contains the interesting part of the page and creates the user account and shows the results. The ASP section is described in detail below. The form section navigates the browser to the first page of the website, CreateAccount.asp when the user clicks the **Next user** submit button.

Website page line	Description
The user account created:	<b>This is not part yet of the ASP section that starts with the &lt;% characters. The text shown is simply copied to the HTML output and presented in the browser.</b>

<%	<b>Start of the ASP section. The lines that follow are ASP specified and terminated with the characters sequence %&gt;.</b>
----	---

```
Set Umra =  
Server.CreateObject("UMRAcom.Umra")
```

The UMRA COM object is now created. The CreateObject method is a member of the ASP built-in Server object. The Server object is available in all ASP pages and offers a number of useful methods (function call). The Server.CreateObject method is the standard method in ASP to initiate an instance of a registered COM object. The argument of the method is the name of the library that holds the COM object (UMRAcom) and the name of the object (Umra). The Set Umra = ... construction creates an ASP script variable with name Umra and sets the value of the variable equal to the result of the Server.CreateObject method: the COM object. In the sequel of the ASP script, the COM object can now be used and must be referenced as the variable name: Umra.

<b>RetVal=Umra.Connect("AMAZONE",56814)</b>	<b>The Connect method is part of the interface of the Umra COM object. It is used to setup a connection between the COM object itself and an UMRA Server. The method takes 2 arguments: the name of the computer and the port number used by the UMRA Service. The return value RetVal should be 0 on success and can be used for error handling purposes.</b>
---	--

<p><b>Umra.SetVariableText</b> "%FirstName%",Request.Form("FirstName")</p>	<p>The method <b>SetVariableText</b> of the <b>Umra COM</b> object is used to initialize a variable-value pair. The <b>COM</b> object can hold a (single) list with multiple variable-value pairs. This list is sent to the <b>UMRA Service</b> when a project is executed. The <b>SetVariableText</b> method takes two arguments: the name of the variable (<b>%FirstName%</b>) and the value of the argument. In this case, the value is copied from the specified input field using the <b>ASP Request</b> object. The <b>Form("FirstName")</b> phrase refers to the input field with name <b>FirstName</b> of the original form of page <b>CreateAccount.asp</b>: <pre>&lt;INPUT TYPE="TEXT" NAME="FirstName"&gt;</pre>When this line of the script is executed, the list with variables stored in the <b>Umra COM</b> object holds the new variable-value pair.</p>
--	--



<b>Umra.SetVariableText</b> <b>"%LastName%",Request.Form("LastName")</b>	<b>Another variable-value pair is added to the list maintained by the UMRA COM object: The last name of the user account specified by the end-user in the form of website page CreateAccount.asp is stored as variable %LastName% and copied from the input text field with name LastName.</b>
---	--

```
RetVal=Umra.ExecuteProjectScript("CreateAccount")
```

The interface member `ExecuteProjectScript` of the COM object is now used to execute the project script on the UMRA Service. The only argument of the member function is the name of the project. The current variable list stored in the COM object is used as the input of the form project. Note that the project script is not executed by the UMRA COM object. Instead, the UMRA COM object instructs the connected UMRA Service to execute the project. The `RetVal` numerical variable returns as zero on success. The value can be used for error handling. When the project script is executed successfully, the variable list of the COM object is updated with the values that are generated by the UMRA project script.

<b>RetVal=Umra.GetVariableText("%UserName%",UserName)</b>	The <b>GetVariableText</b> interface member function is used to obtain the text value of the specified variable. The variable should be part of the variable list of the UMRA COM object. On success, the RetVal value should be zero and the ASP script variable <b>UserName</b> is filled with the actual value of the variable.
<b>Response.Write "&lt;BR&gt;"</b>	A break is written to the output HTML sequence that is generated by this ASP page.
<b>Response.Write "User name: "</b>	The text <b>User name:</b> is written to the HTML output.
<b>Response.Write UserName</b>	The value of the ASP variable <b>UserName</b> , as collected from UMRA variable <b>%UserName%</b> , written to the HTML output.

<code>RetVal=Umra.GetVariableText("%Password%",Password)</code>	The value of UMRA variable %Password% is searched for in the list maintained by the UMRA COM object. When found, the value is stored in ASP variable Password.
<code>Response.Write "&lt;BR&gt;"</code>	A break is written to the HTML output.
<code>Response.Write "Password: "</code>	The text Password: is written to the HTML output.
<code>Response.Write Password</code>	The value of the password is written to the HTML output.
<code>%&gt;</code>	Termination of the ASP section. Normal HTML code follows after this character sequence or a new ASP section can start. In this example project, a small form is shown to navigate to the first page of the website.

Table 3: Detailed description of the ASP page section using UMRA COM objects.

### Testing the website

The UMRA application is now ready for testing. Start Windows Internet Explorer and connect to the IIS website by entering the URL address of the first web page:

*<http://amazone:81/CreateAccount.asp> <http://amazone:81/createaccount.asp>*

Here, **amazone** is the name of the computer that runs IIS. The port for the website is configured as 81. When automatic logon is enabled, the user (you) is authenticated by IIS and the HTML code of the ASP page is shown. If automatic logon is not enabled, you need to logon first.



Figure 14: First page of the UMRA website on IIS.

Enter the first and last name of a user account and click the **Create Account** button. The form is submitted to IIS and the ASP page **ShowResults.asp** is executed.

As part of the ASP page, the **UMRA Service** is contacted and requested to execute the project script that creates the account. Results are returned by the service and shown in the web page.



Figure 15: Result page of the UMRA website.

You can check the **UMRA Service** log file for progress and debugging information. The log file **UmraSvcLogX.txt** can be found in the Log directory of the **UmraService** program directory.

#### UMRA Service log file

The COM object connects to the **UMRA Service** and the variables are listed in the log file. The **%UmraFormSubmitAccount%** variable shows the end-user of rh browser.

The log file contains the following section upon completion of a successful session:

```
11:16:53 01/03/2006 Variable 1: %FirstName%=John
11:16:53 01/03/2006 Variable 2: %LastName%=Williams
11:16:53 01/03/2006 Variable 3:
%UmraFormSubmitAccount%=T4ELOC2\Administrator
11:16:53 01/03/2006 Creating AD account in specified domain:
'tools4ever.local2'.
```

11:16:53 01/03/2006 Creating AD account in container 'Users'.

11:16:53 01/03/2006 Creating AD account in Organizational Unit-Container:  
'LDAP://CN=Users,DC=tools4ever,DC=local2'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2': Using name generation algorithm  
'Default', 100 iterations maximum for duplicate names.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Common name of user set to  
'John Williams'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. SAM account name (username) of  
user set to 'williamsj'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute  
'userPrincipalName' of object 'John Williams' set to  
'williamsj@tools4ever.local2'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute 'displayName' of  
object 'John Williams' set to 'John Williams'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute 'givenName' of  
object 'John Williams' set to 'John'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. LDAP attribute 'sn' of object 'John  
Williams' set to 'Williams'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Password generated ('Strong, 7  
chars with 1 numeric, 1 special (variable: %Password%)') and set in variable  
'%Password%'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Account  
disabled'=FALSE (101034). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Password  
never expires'=FALSE (101032). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Store password using reversible encryption'=FALSE (101033). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Smart card is required for interactive logon'=FALSE (101035). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Account is trusted for delegation'=FALSE (101036). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Account is sensitive and cannot be delegated'=FALSE (101037). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Use DES encryption types for this account'=FALSE (101038). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Boolean parameter 'Don't require Kerberos preauthentication'=FALSE (101039). Result not changed.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Account expiration date not specified for new user 'John Williams' object.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. User 'John Williams' successfully created.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Password set for new user object 'John Williams'.

11:16:53 01/03/2006 Creating AD user account in container/OU  
'LDAP://CN=Users,DC=tools4ever,DC=local2'. Flag 'User Cannot Change Password' not changed from default value (FALSE).

11:16:53 01/03/2006 Form message:  
'01/03/2006,11:16:53,T4ELOC2\Administrator,"Automation run script",OK,CreateAccount'



## **UMRA COM on 64-bit platforms**

### **Current version of UMRA**

The UMRA COM DLL can be used on both 32-bit and 64-bit platforms running IIS. The UMRA COM software is available for both platforms and installed automatically as part of the UMRA Automation module. For the different platforms, UMRA uses the following files:

<b>Platform:</b>	Intel 32-bit platforms (w32)
<b>Default program directory:</b>	C:\Program Files (x86)\Tools4ever\User Management Resource Administrator
<b>UMRA COM files:</b>	UmraCom.dll JvrLog.dll

<b>Platform:</b>	AMD based 64-bit platforms (x64)
<b>Default program directory:</b>	C:\Program Files (x86)\Tools4ever\User Management Resource Administrator
<b>UMRA COM files:</b>	UmraCom64.dll JvrLog64.dll

On 64-bit platforms running IIS, a web page automatically uses either the 32-bit or 64-bit version of the UMRA COM object. This depends on the exact configuration of IIS. See knowledge base article KB894435 for more information.

### **Older version of UMRA**

In older versions of UMRA, the most recent version being 9.0 build 1425, the UMRA COM DLL was available only as a 32-bit COM DLL. Microsoft supports 32-bit COM DLL's to be used on 64-bit IIS platforms by configuring IIS. In a 64-bit environment, IIS can be configured to run ASP pages using 32-bit COM DLL's.

### **Procedure**

1. Install UMRA, including UMRA automation on the 64-bit computer that runs IIS. This will install and register the 32-bit COM object;
2. Open a command prompt and navigate to the %systemdrive%\Inetpub\AdminScripts directory. By default, %systemdrive% equals "C:".
3. Type the following command: **cscript.exe adsutil.vbs set W3SVC/AppPools/Enable32BitAppOnWin64 "true"**
4. Press ENTER. Something similar to the following is shown:

Microsoft (R) Windows Script Host Version 5.6

Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Enable32BitAppOnWin64 : (BOOLEAN) True

The setting is now complete. From now on, ASP pages can successfully use the UMRA COM object.

In case the setting is not made, and an ASP page using the UMRA COM object is executed, the following error may appear:

```
...
Server object error 'ASP 0196 : 80040154'
Cannot launch out of process component
/ShowResults.asp, line 8
Only InProc server components should be used. If you want to use LocalServer
components, you must set the AspAllowOutOfProcComponents metabase
setting. Please consult the help file for important considerations.
...
```

### **IIS configuration Windows Server 2008**

By default, ASP is not enabled for IIS running on Windows Server 2008. To enable ASP, use the following procedure on Windows Server 2008:

1. Select **All Programs, Administrative Tools, Server Manager**;
2. If this is not the case, add the **Web Server (IIS)** role to the server;
3. In the **Server Manager**, select **Web Server (IIS)** and navigate to the section of **Role Services**.
4. Click **Add Role Services** and select **Web Server, Application Development, ASP**.
5. Complete the procedure.

Now, a web-site can use the UMRA COM object in ASP pages.

### **3.3.5. References**

*Beginning ATL COM Programming*, by Richard Grimes, Alex Stockton, George Reilly and Julian Templeman. Wrox Press Ltd, 1998

*DCOM – Microsoft Distributed Component Object Model*, by Frank E. Redmond  
III. IDG Books, 1997



---

### 3.4. Managing printer queues

With User Management Resource Administrator (UMRA) you can let the helpdesk manage printer queues and print jobs. In this document, a sample project is described to manage a Windows printer queue.



*Read the full PDF version of UMRA Managing Printer queues*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/Print-Job-Management.pdf>

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### 3.4.1. Introduction

With User Management Resource Administrator (UMRA) you can let the helpdesk manage printer queues and print jobs. Individual print jobs can be paused, restarted, resumed and deleted. The printer spooler service itself can be restarted.

This example project shows you how to setup UMRA form projects to implement this type of functionality.

In the example project, a form shows the printer documents of a particular printer. From the list, a document can be selected. The user can then press a button to pause, restart, resume or delete the print job.

#### Printer queue management - Form result

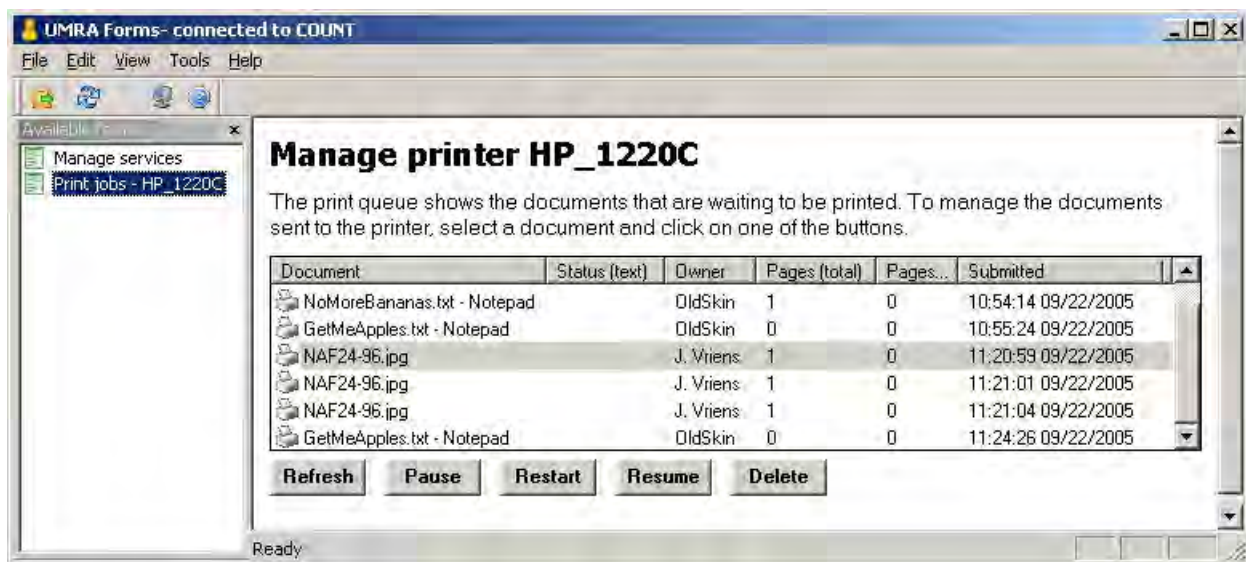


Figure - UMRA Forms application running the example project

This document describes the main implementation aspects of this example project. The example project is also available from the Tools4ever web-site.

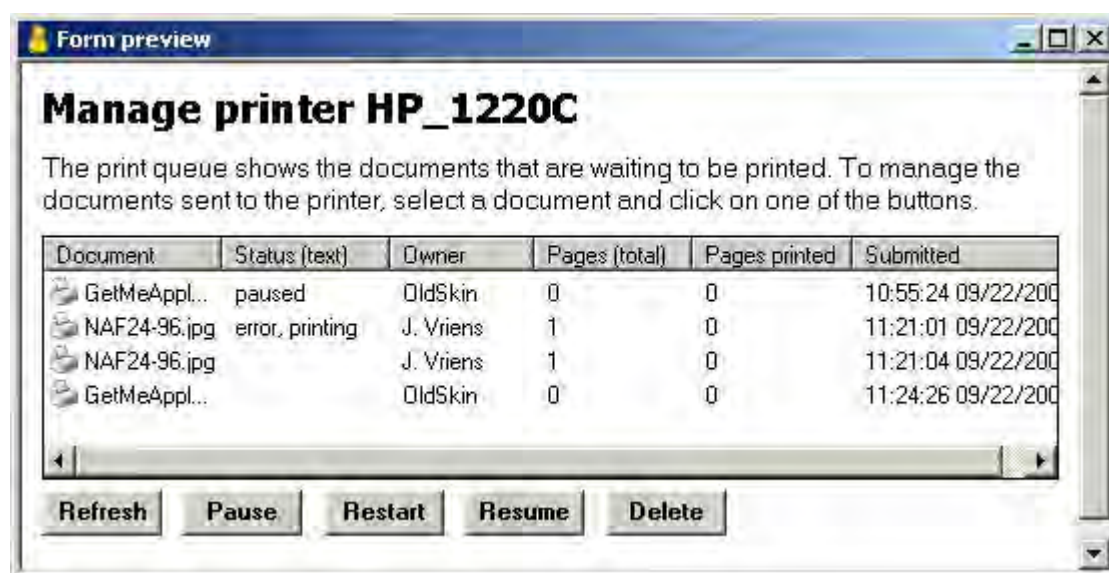
The reader of this document is supposed to be familiar with the **UMRA Console**, **Forms** and **Service** applications and the basics of **UMRA Form** projects.

### 3.4.2. UMRA projects for managing printer queues

#### Project description

The example project consists of 2 UMRA forms project:

**Main project - Print jobs – HP\_1220C:** The main project that contains the form and the script that is executed when one of the buttons is pressed.



**Form preview**

### Manage printer HP\_1220C

The print queue shows the documents that are waiting to be printed. To manage the documents sent to the printer, select a document and click on one of the buttons.

Document	Status (text)	Owner	Pages (total)	Pages printed	Submitted
GetMeAppl...	paused	OldSkin	0	0	10:55:24 09/22/200
NAF24-96.jpg	error, printing	J. Vriens	1	0	11:21:01 09/22/200
NAF24-96.jpg		J. Vriens	1	0	11:21:04 09/22/200
GetMeAppl...		OldSkin	0	0	11:24:26 09/22/200

Refresh Pause Restart Resume Delete

Figure 1 – Form of project Print jobs – HP\_1220C

The form contains some introduction static text fields, the main table showing print documents and the submit buttons.

**Auxiliary project - Print job list – HP\_1220C:** A simple project used to collect the printer documents and store the information in a table variable.

#### Project principle

The main project **Print jobs – HP\_1220C** shows the table with printer documents. The table data is obtained from a variable that is generated in the other project **Print job list – HP\_1220C**. This project is set as the initial project of the main project.

When the user selects a document from the table, the **DocumentID** of the document is stored in a variable. Note that this **DocumentID** field is part of the table but shown in a column with a width of 0%.

Next, the user presses one of the buttons. A variable is set, referring to the button pressed and this information, together with the **DocumentID** is sent to the UMRA Service.

The **UMRA Service** processes and executes the command, collects the new printer document information and returns the same form.



### Auxiliary project - Print job list - HP\_1220C

The helper project only contains a script, no form. The script is executed as the initial project of main project **Print jobs – HP\_1220C**. The script only contains 2 lines.

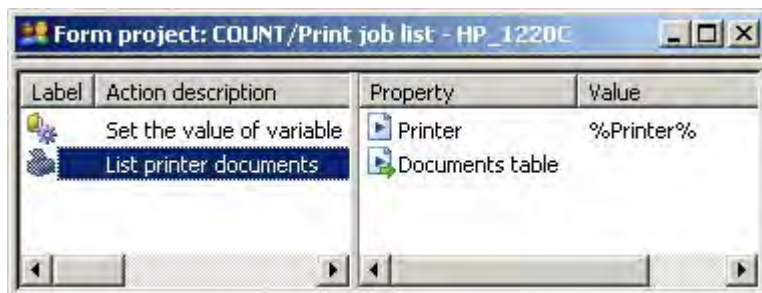
The first script action sets the value of variable **%Printer%**. The value must equal the name of a network printer (queue). Syntax of this value is: `\\name_of_computer\name_of_printer`. Example: `\\COUNT\HP DeskJet 1220C`.



*Figure 2 – Setting the variable %Printer% to the required printer*

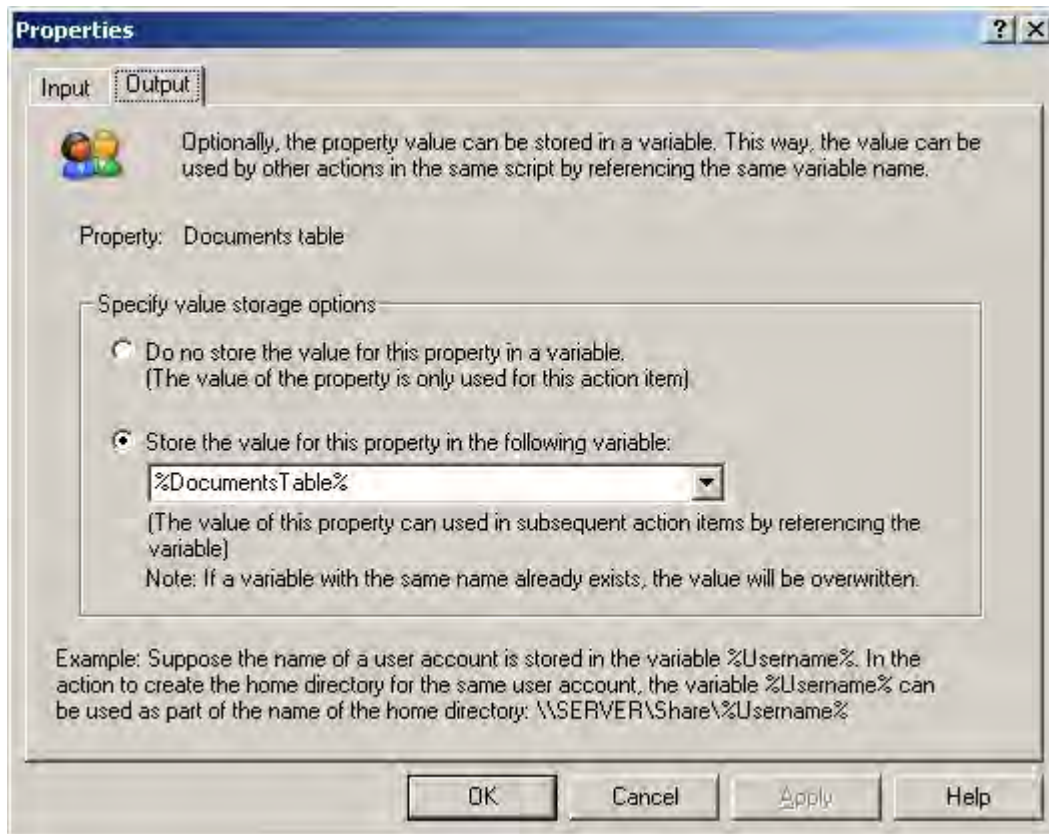
In another network environment, this variable must be set to the name of the printer of interest. The variable is used in the other script action and in the other main form project.

The second action collects the list with printer documents.



*Figure 3 - Action - Get printer documents info and store in table variable*

The action collects the list with printer documents from the specified **%Printer%**. The result is a table that is stored in a variable specified for property **Documents table**.



*Figure 4 – Storing a table with documents info in variable %DocumentsTable%*

The property is an output only property: the property is not needed as an input value to execute the action. Instead, the property is used to hold the output values of the action.

#### **Print jobs project - Form**

The form of the main project **Print jobs – HP\_1220C** contains some introduction static text fields, the main table showing print documents and the submit buttons.

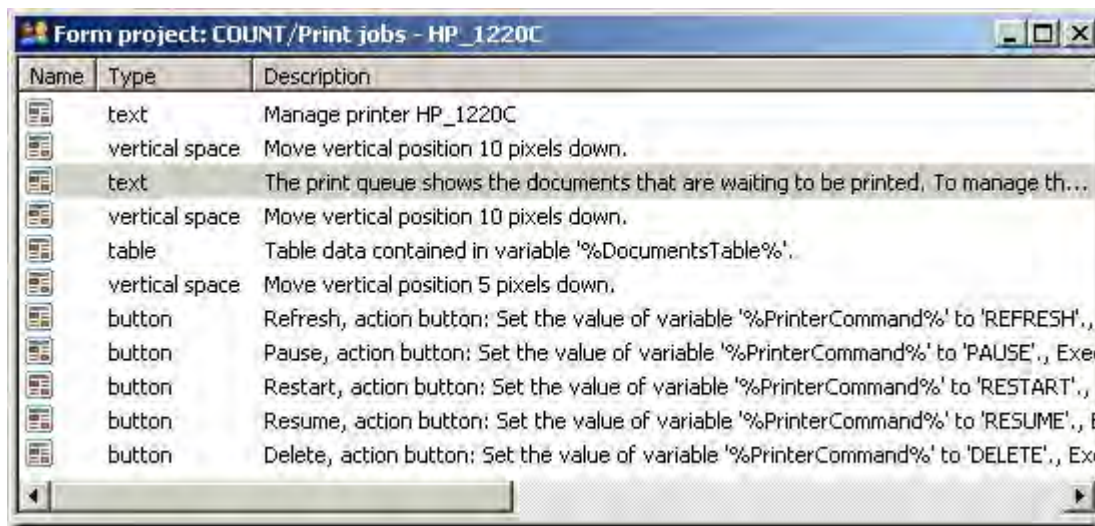
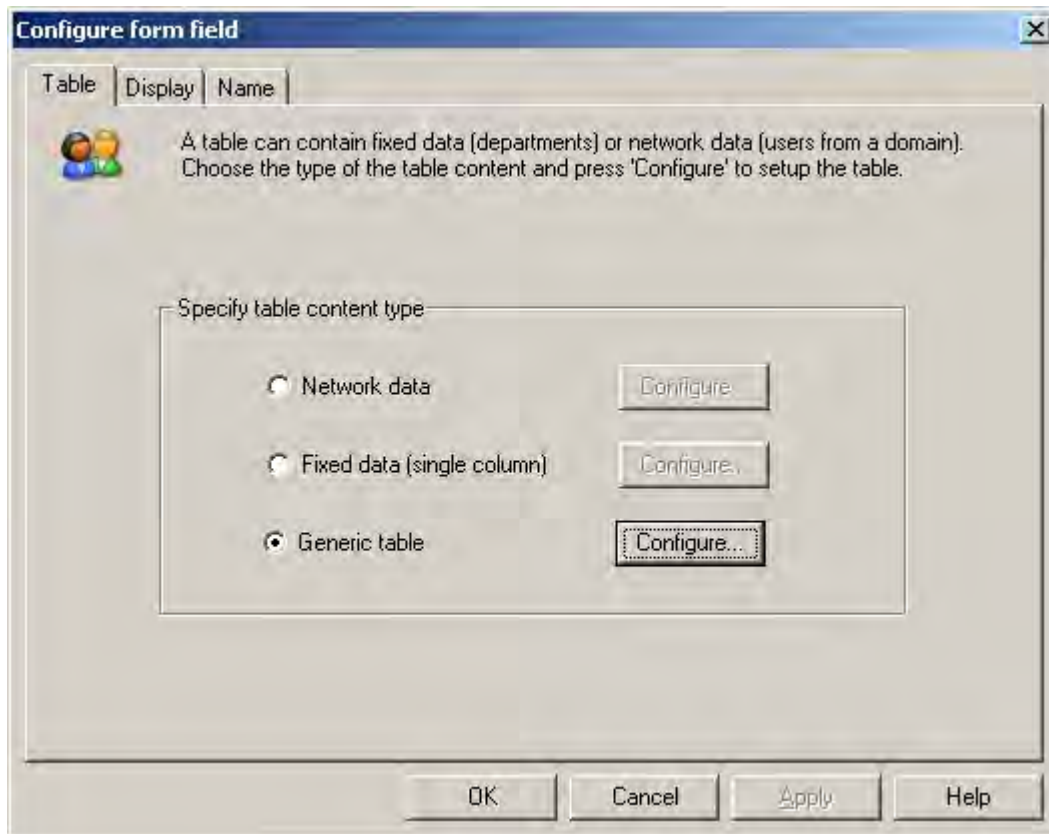


Figure 5 - Designing the main form

Except for the table and buttons, the fields are easy to implement and customize. The table and buttons fields are explained in the next topics.

#### Print jobs project - Table with printer documents

The table with printer documents is configured as a **generic table**, specified by a variable.



*Figure 6 - Table form field configured as Generic table*

Click **Configure...** . In the **Configure table** window, click **Configure**.

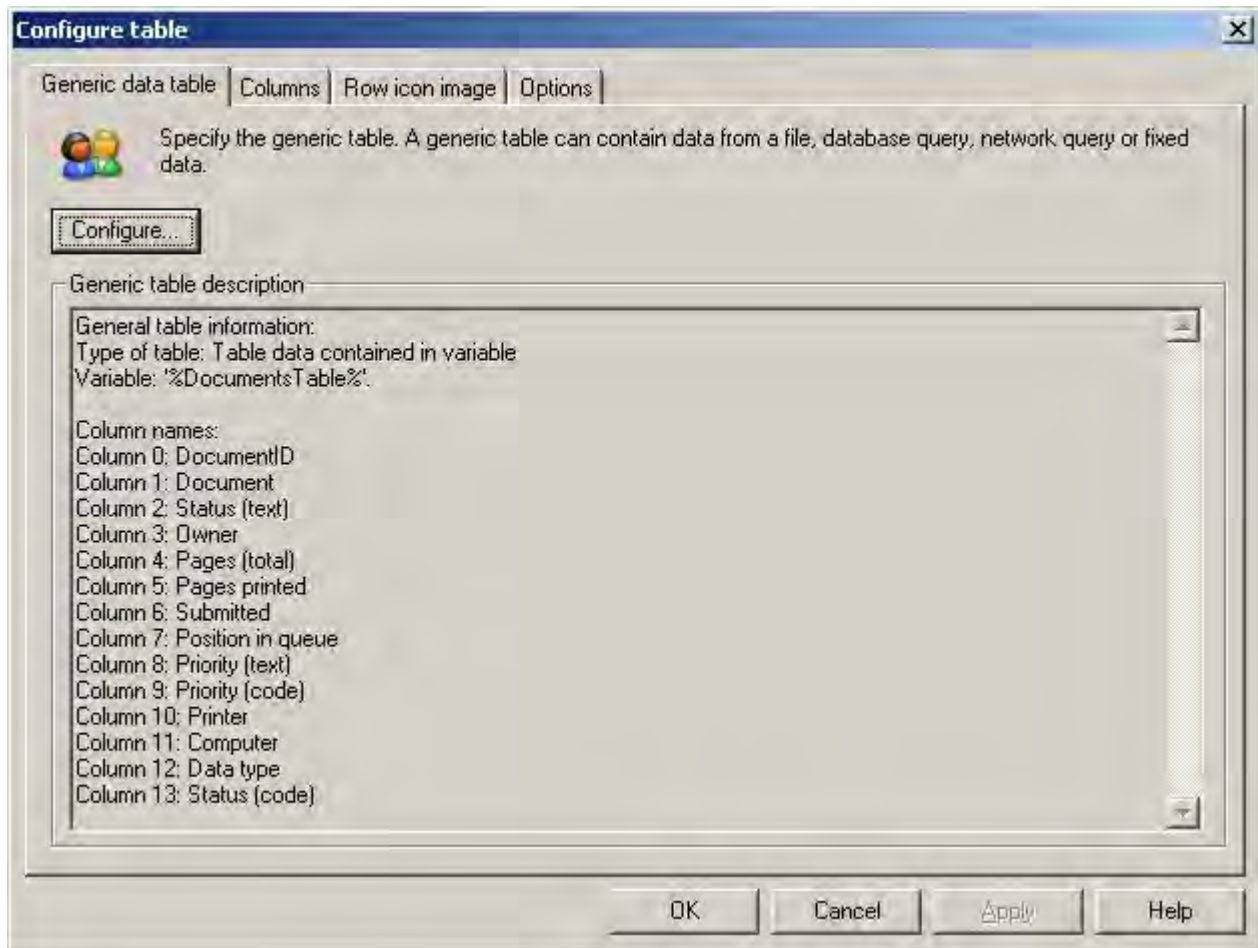


Figure 7 – Configuring the table



Select the table type as **Variable** and click the tab **Variable generic table**.

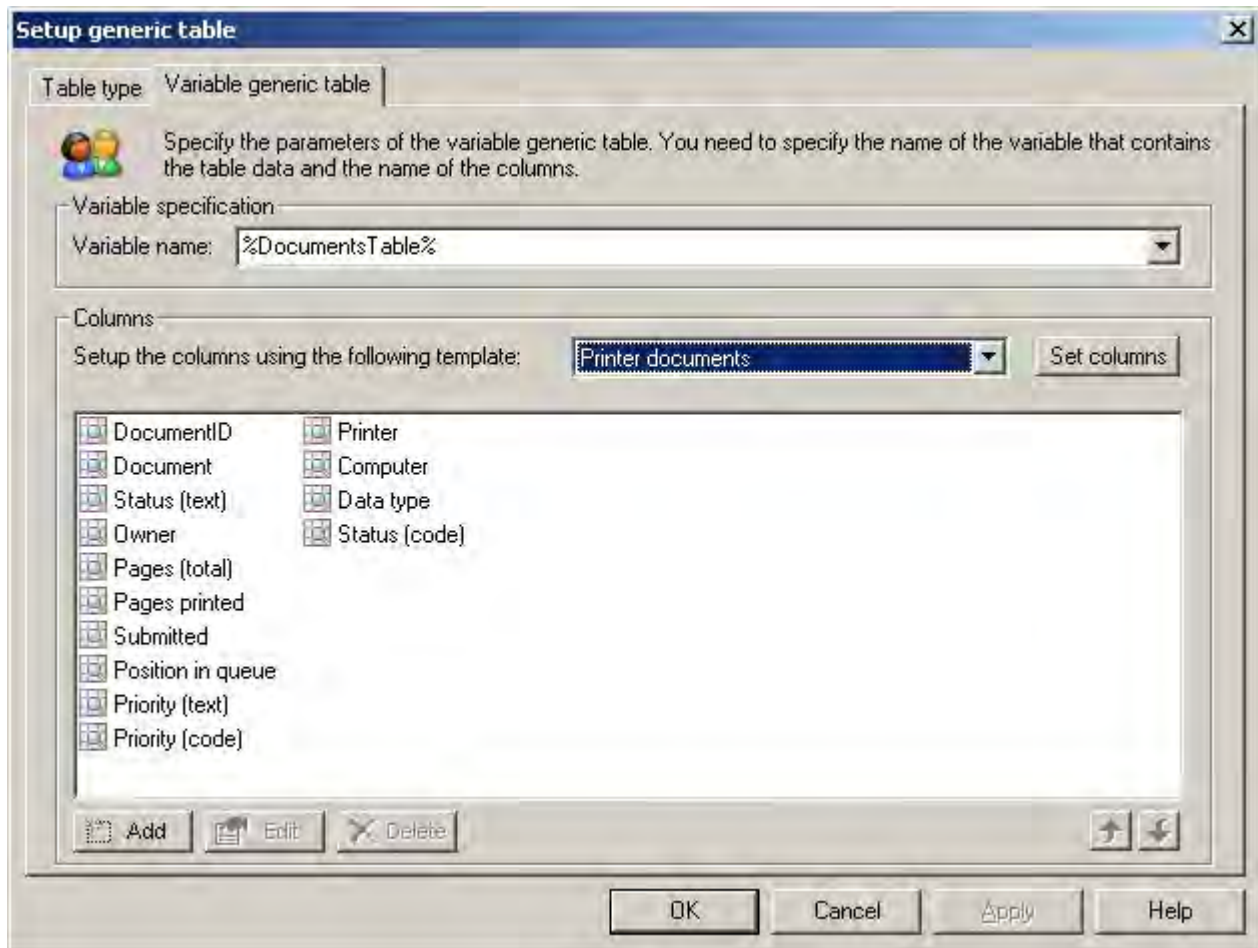


Figure 8 - Setup variable generic table

#### Table variable specification

The window is used to specify the name of the variable that holds the table data and to setup the columns of the table. The **variable** name must be equal to the name of the variable generated by the auxiliary project: **%DocumentsTable%**. Note that this project has no knowledge of the columns that exist for the table. That's why you need to specify the names of the columns.

#### Table columns specification

The table data is generated by the action **List printer documents** of helper project **Print job list – HP\_1220C**. Since we know that, we know what columns are contained by the resulting table data variable.

1. Since the action always generates the same columns, you can select the template **Printer documents** and press **Set columns** to setup the column configuration.
2. Press **OK** to continue.

3. To set up the columns that are shown in the form, select tab **Columns** in the window **Configure table**.
4. The list with **Available columns** shows the columns as configured in step 3. In the list **Current column configuration** you need to setup the column that must be shown in the form.

Note that this column information is only used to give a meaning to the columns of table data variable **%DocumentsTable%**. The columns do not necessarily correspond with the columns shown in the form.

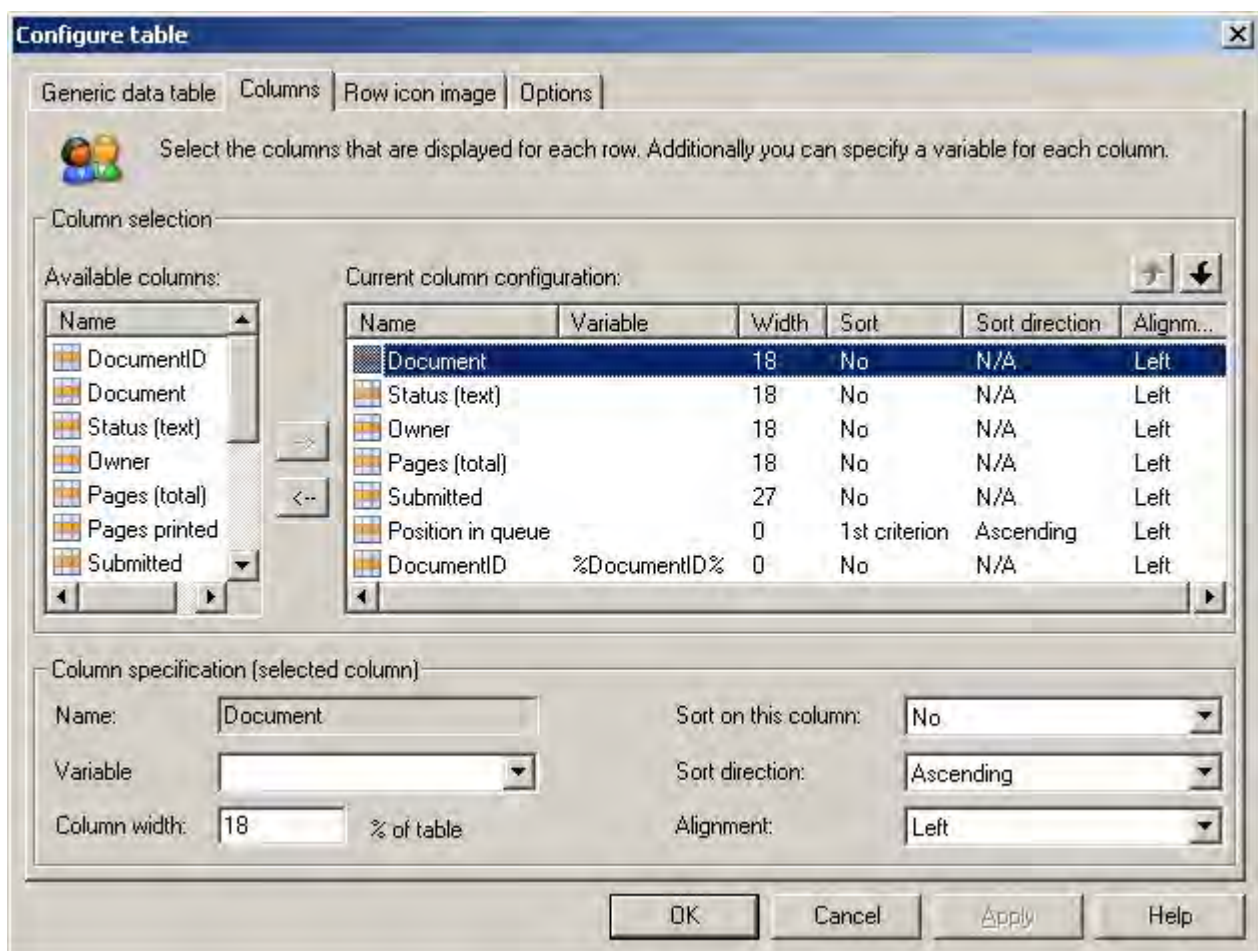


Figure 9 – Setting up columns for the form table and variable **%DocumentID%**

#### Form table return variable

This window is also used to setup the variable that is used when the end-user selects a document from the table. This variable is used to identify the printer document that must be sent a command. The action to execute a printer command needs the id number of the document as an input property. Therefore, the variable **%DocumentID%** is configured for column **DocumentID**. Since this information is not valuable to the user, the width of the column **DocumentID** is set to 0%.

Now when the end-user selects a document from the list, the **DocumentID** number is stored in variable **%DocumentID%** and sent to the UMRA Service for further processing.

Use the **Row icon image** and **Options** tab to fine tune the table form field with printer documents.

### Print jobs project - Form buttons

The form button fields have a similar configuration. The buttons show the command to be executed:

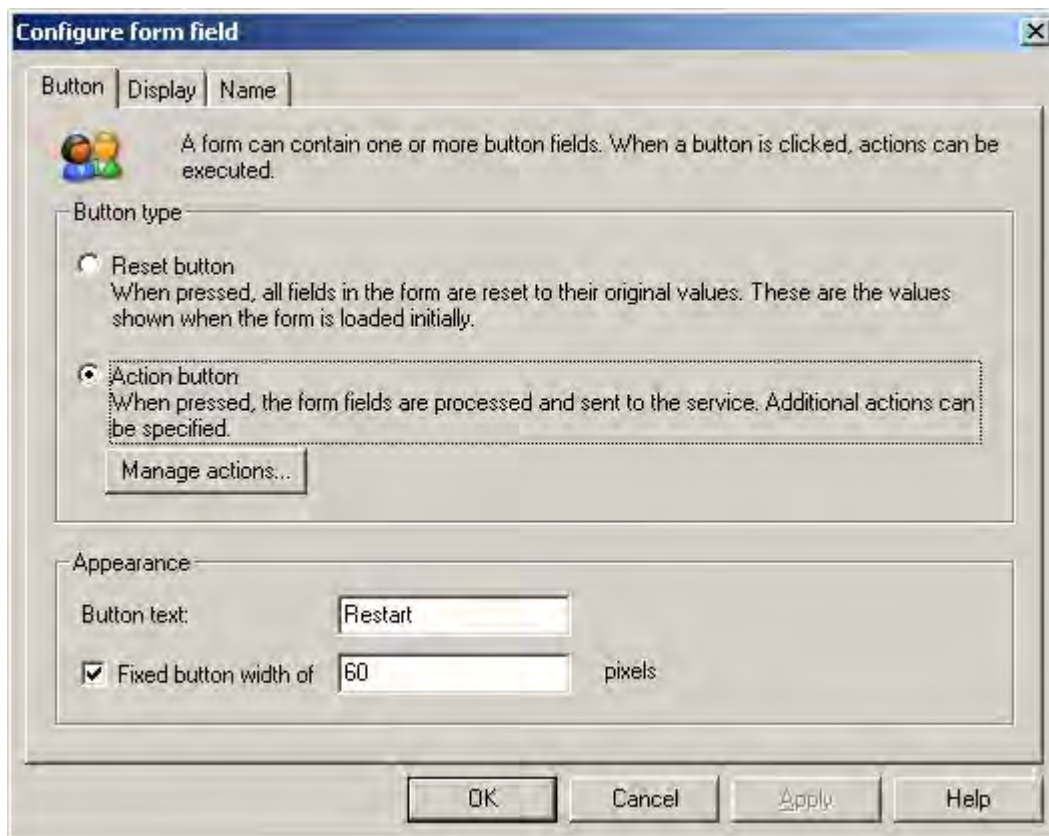


Figure 10 - Button form field configuration, setup **Button text**

For each button, 3 actions are configured:



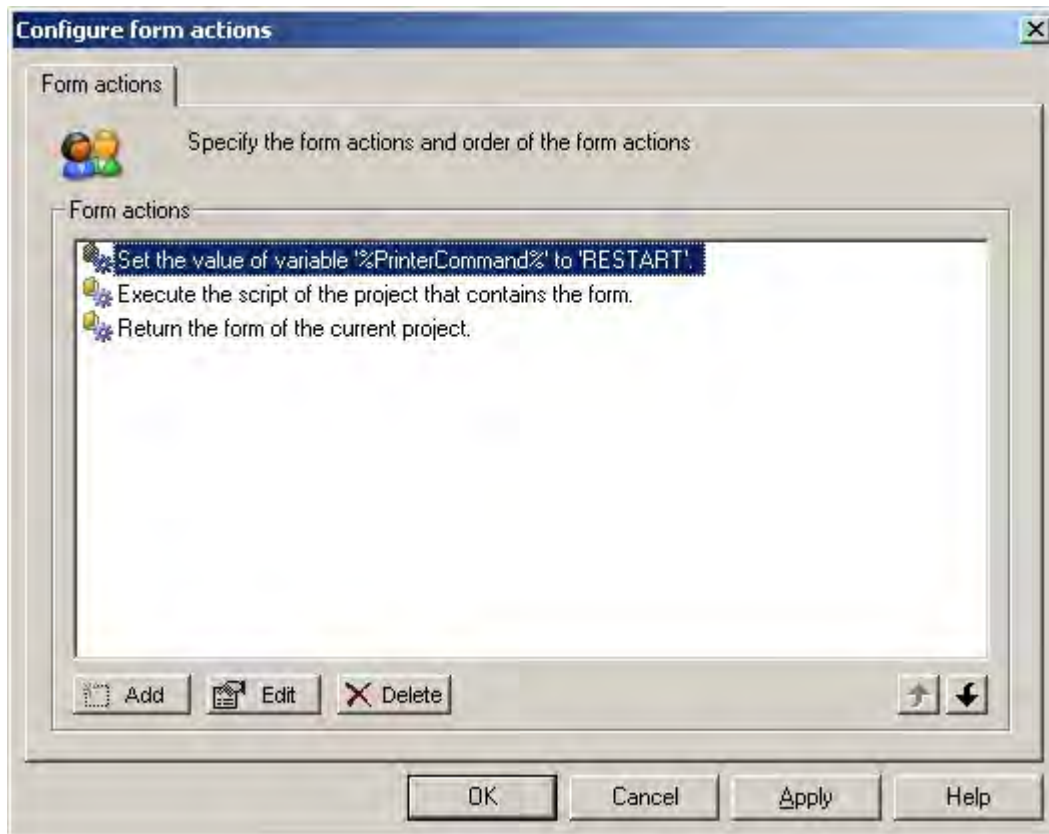


Figure 11 – Button actions

Set the value of variable **%PrinterCommand%**: The variable **%PrinterCommand%** is sent to the UMRA Service when the button is pressed. The variable is used in the script of the project to execute the appropriate actions.

1. **Execute the script of the project that contains the form**: The script is executed to send the printer a printer document command.
2. **Return the form of the current project**: When the printer is sent a command, the same form is returned, to reflect the status of the command sent and to allow the user to issue a command for another printer document.

#### Print jobs project - Script

The script of the main project **Print jobs – HP\_1220C** first performs some error handling to check the end-user input. Next, control is passed to the correct action that corresponds with the button pressed.

### Script action 1: Check %DocumentID%

In the first action, the variable %DocumentID% is tested to see if a document was selected from the list. If the end-user presses a submit button and no document is selected, the value either does not exist or equals zero.

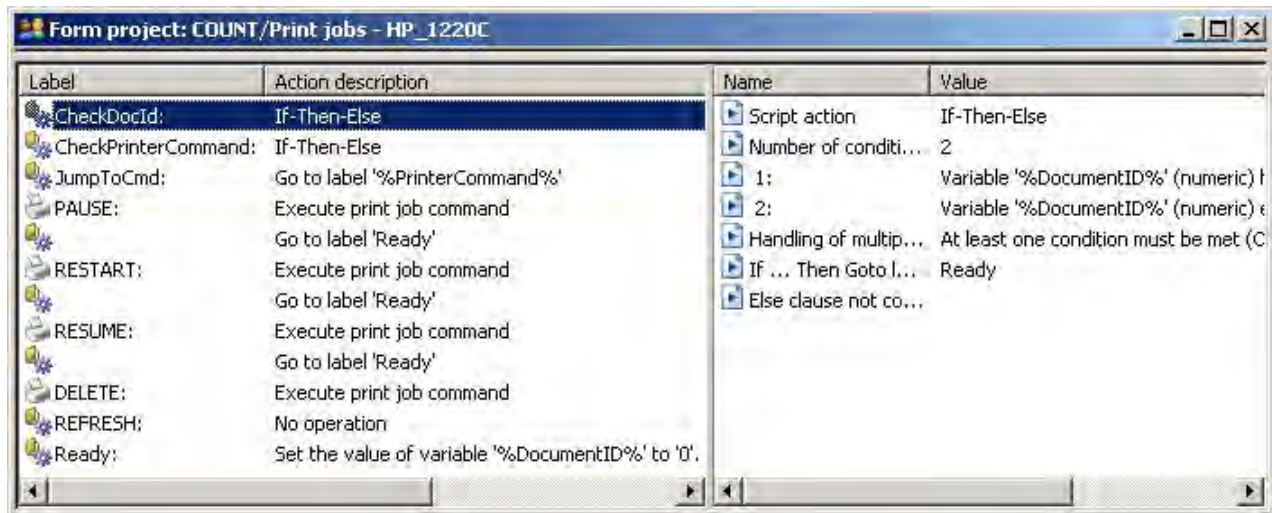


Figure 12 - Script action to check if a document was selected

The configuration of the **If-Then-Else** script action to check the value is shown below:

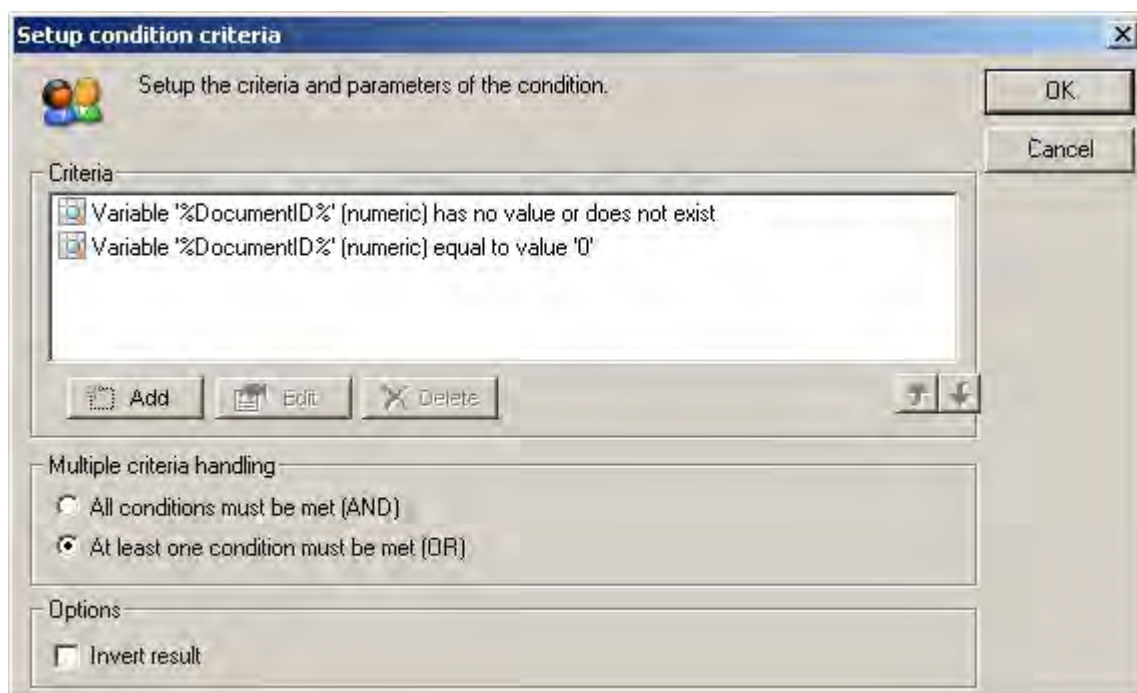


Figure 13 - Configuration of the If-Then-Else conditions to check the %DocumentID% variable

When the value is not valid, script action execution continues with the action with label **Ready**.

*Script action 2: Check %PrinterCommand%*

The next action checks the input printer command %PrinterCommand%. Theoretically, this action is not necessary, but it is a good habit to include thorough error-handling.

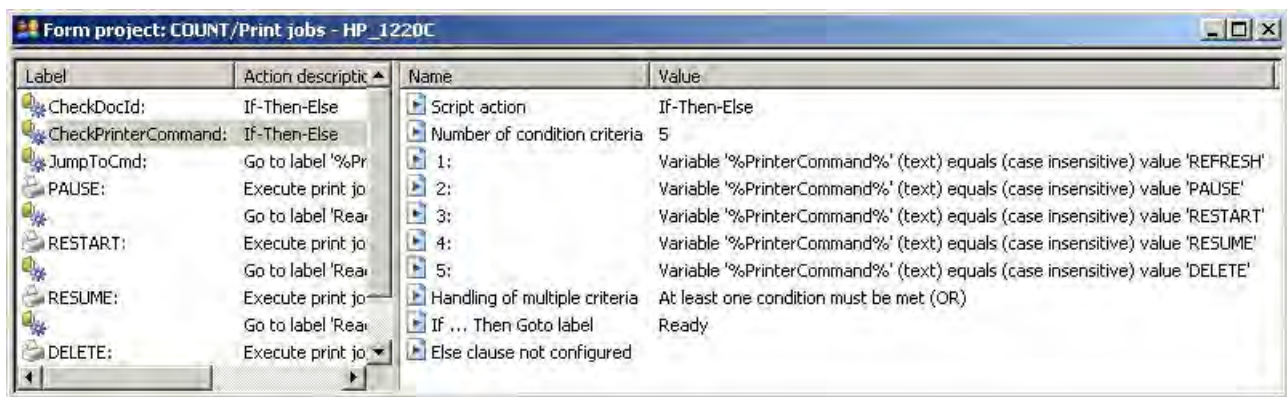


Figure 14 - Script action to check input variable %PrinterCommand%

The action checks the value of variable %PrinterCommand%. If the value is not specified or does not contain one of the valid options, script action execution continues with the action with label **Ready**.

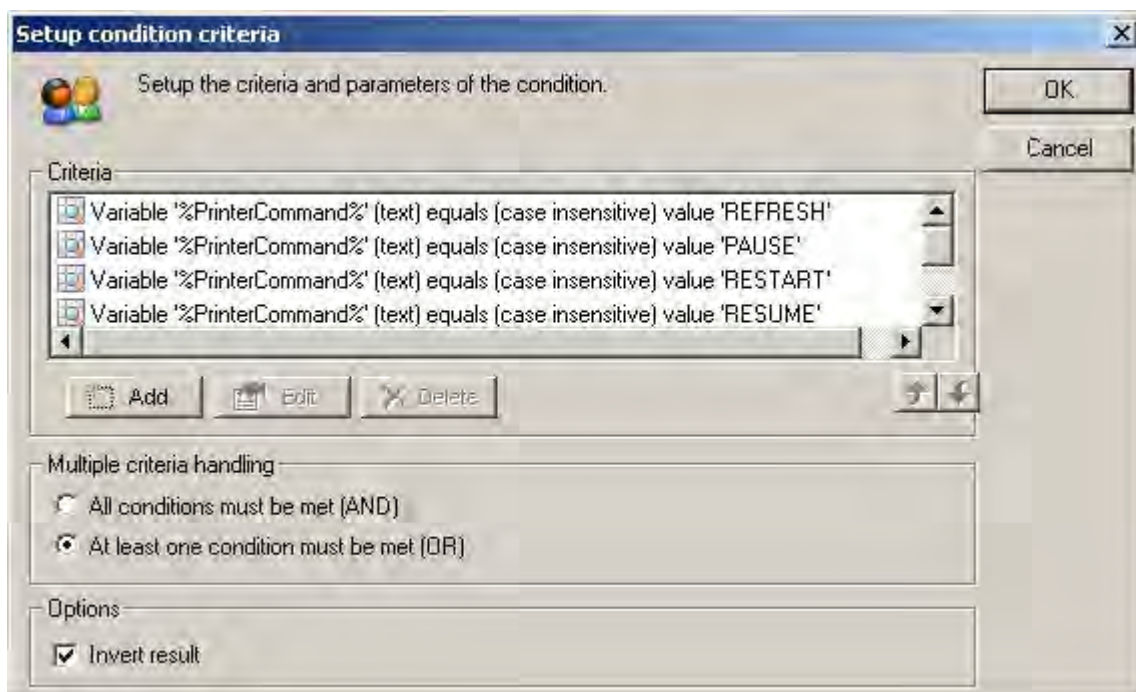


Figure 15 - Configuration of the If-Then-Else conditions to check the %PrinterCommand% variable

*Script action 3: Go-To printer command*

When the input variables are checked, the script action **Go to label %PrinterCommand%** is used to continue script action execution with the action **Execute print job command** that corresponds with the button pressed.

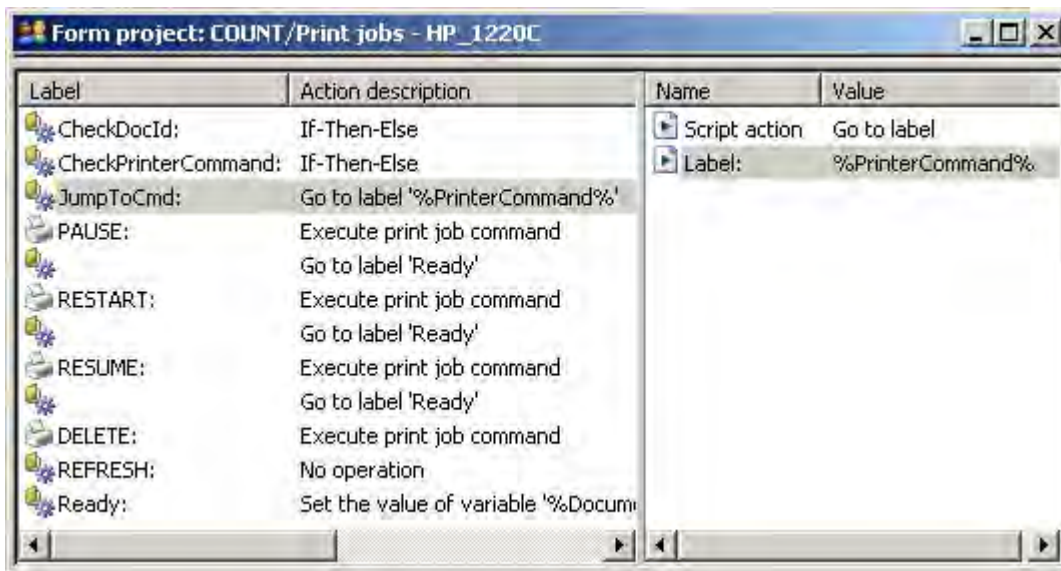


Figure 16 - Go to label %PrinterCommand% action

Now how does this work? For each of the possible printer commands, the script contains a separate section containing two script actions. The first action executes the specific printer command. The printer execution action contains a label so that the action can be jumped to. The name of the label corresponds with the value of the variable **%PrinterCommand%**. This value is set as a form action when the end user presses one of the form submit buttons. When the printer execution action is executed, the **Go to label 'Ready'** action is used to end the script.

Example: When the user presses the **Restart** button in the form, the variable **%PrinterCommand%** is set to **RESTART**. When the script is executed, the action **Go to label %PrinterCommand%** is executed as: **Go to label RESTART**. The **RESTART** label action executes the printer command to restart the printer document.



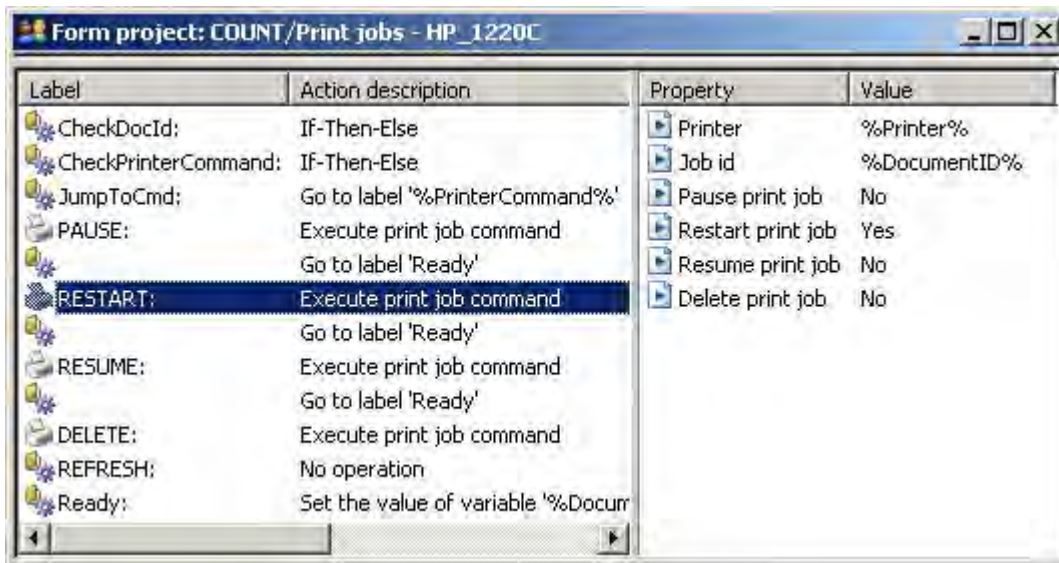


Figure 17 – Executing the print job command action

*Script action 4: Execute print job command*

The execute print job command, UMRA accesses the printer **%Printer%** as specified with helper project **Print job list – HP\_1220C** and job **%DocumentID%**. This printer document ID number corresponds with the ID of the document that is selected from the table in the form.

For the **Restart** action, the property **Restart print job** is set to **Yes**. The next action jumps to label **Ready** to end the script.

When the script is completed, the next form button action is executed: **Return the form of the current project**: The cycle starts over again. As part of the form generation process, the script of the initial project **Print job list – HP\_1220C** is executed.

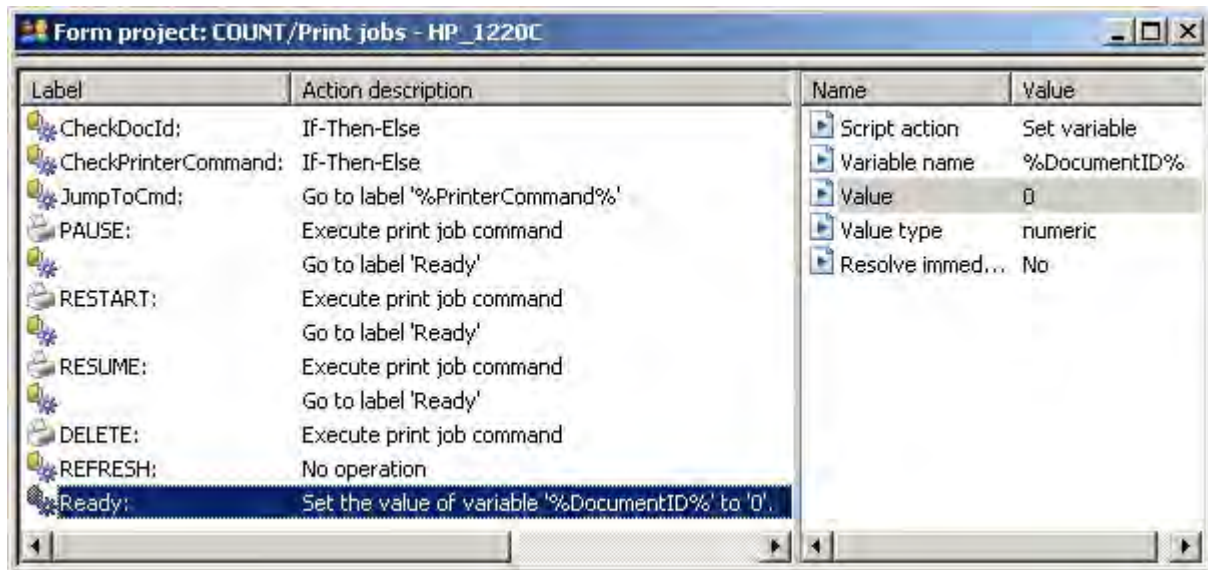


Figure 18 – Resetting the %DocumentID% variable to 0

#### Script action 5: Reset %DocumentID%

To re-initialize the %DocumentID% variable, the last action sets the value of the variable to 0. This ensures that no document is referred to when the script is executed the next time in case the end user has not selected a document.

#### Linking the auxiliary project to the main project

The auxiliary project **Print job list – HP\_1220C** retrieves the printer documents information and stores this table information in variable %DocumentsTable%. The main project shows the printer documents in a table of the form of the project and sends the printer a command when a button of the form is pressed.

#### Initial project specification

Now how do the projects work together? The helper project is set as the initial project of the main project. When the form of the main project is generated by the UMRA Service, the script of the project **Print job list – HP\_1220C** is executed first.

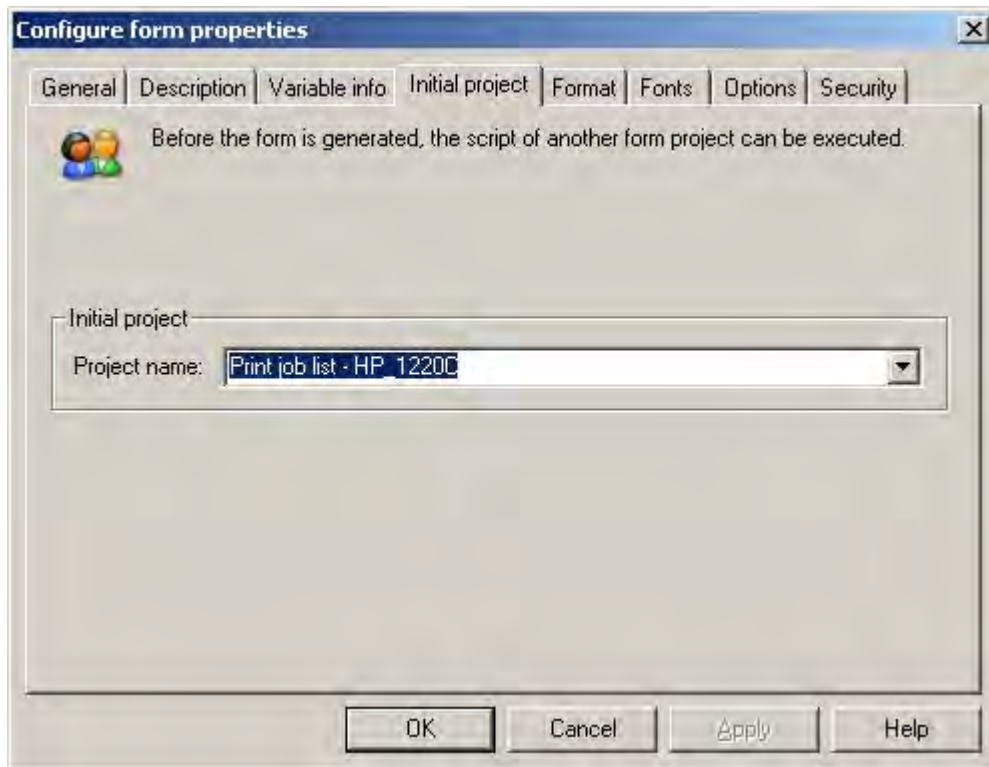


Figure 19 - Initial project configuration of project *Print jobs – HP\_1220C*

The helper project **Print job list – HP\_1220C** collects the printer document information and stores the resulting table in variable `%DocumentTable%`. This variable is passed to the main project and used in the main table form field.

### Project execution

The following section shows the log of the **UMRA Service** when the **UMRA Forms** application connects to the service, selects the form, selects a document and presses one of the submit buttons.

### Project execution log

09:25:49 09/23/2005 Form message: '09/23/2005,09:25:49,"SSP\J. Vriens","Forms list",OK,N/A,"2 projects found for user 'SSP\J. Vriens'.'"

09:25:53 09/23/2005 Executing form initialization project 'Print job list - HP\_1220C'.

09:25:53 09/23/2005 Variable 1: %UmraFormSubmitAccount%=SSP\J. Vriens

09:25:53 09/23/2005 List documents in printer (queue) '\\COUNT\HP Deskjet 1220C'.

09:25:53 09/23/2005 Form message: '09/23/2005,09:25:53,"SSP\J. Vriens","Form load",OK,"Print jobs - HP\_1220C",'

09:26:04 09/23/2005 Variable 1: %DocumentID%=6

09:26:04 09/23/2005 Variable 2: %UmraFormSubmitAccount%=SSP\J. Vriens

09:26:04 09/23/2005 Variable 3: %Printer%=\COUNT\HP Deskjet 1220C

09:26:04 09/23/2005 Variable 4: %DocumentsTable%=Table with 10 rows

09:26:04 09/23/2005 Variable 5: %NowDay%=23

09:26:04 09/23/2005 Variable 6: %NowMonth%=09

09:26:04 09/23/2005 Variable 7: %NowYear%=2005

09:26:04 09/23/2005 Variable 8: %NowHour%=09

09:26:04 09/23/2005 Variable 9: %NowMinute%=26

09:26:04 09/23/2005 Variable 10: %NowSecond%=04

09:26:04 09/23/2005 Variable 11: %PrinterCommand%=PAUSE

09:26:04 09/23/2005 If-Then-Else condition [Variable '%DocumentID%' (numeric) has no value or does not exist OR Variable '%DocumentID%' (numeric) equal to value '0'] result is FALSE, continue script execution with next action.

09:26:04 09/23/2005 If-Then-Else condition [Variable '%PrinterCommand%' (text) equals (case insensitive) value 'REFRESH' OR Variable '%PrinterCommand%' (text) equals (case insensitive) value 'PAUSE' OR Variable '%PrinterCommand%' (text) equals (case insensitive) value 'RESTART' OR Variable '%PrinterCommand%' (text) equals (case insensitive) value 'RESUME' OR Variable '%PrinterCommand%' (text) equals (case insensitive) value 'DELETE' (invert)] result is FALSE, continue script execution with next action.

09:26:04 09/23/2005 Jump to script action with label 'PAUSE'.

09:26:04 09/23/2005 Executing command 'Pause' for job id '6' of printer (queue) '\COUNT\HP Deskjet 1220C'.

09:26:04 09/23/2005 Command successfully executed

09:26:04 09/23/2005 Jump to script action with label 'Ready'.

09:26:04 09/23/2005 Executing form initialization project 'Print job list - HP\_1220C'.

09:26:04 09/23/2005 Variable 1: %DocumentID%=0

09:26:04 09/23/2005 Variable 2: %UmraFormSubmitAccount%=SSP\J. Vriens

09:26:04 09/23/2005 Variable 3: %Printer%=\COUNT\HP Deskjet 1220C

09:26:04 09/23/2005 Variable 4: %DocumentsTable%=Table with 10 rows

09:26:04 09/23/2005 Variable 5: %NowDay%=23



09:26:04 09/23/2005 Variable 6: %NowMonth%=09

09:26:04 09/23/2005 Variable 7: %NowYear%=2005

09:26:04 09/23/2005 Variable 8: %NowHour%=09

09:26:04 09/23/2005 Variable 9: %NowMinute%=26

09:26:04 09/23/2005 Variable 10: %NowSecond%=04

09:26:04 09/23/2005 Variable 11: %PrinterCommand%=PAUSE

09:26:04 09/23/2005 List documents in printer (queue) '\\COUNT\HP Deskjet 1220C'.

09:26:04 09/23/2005 Form message: '09/23/2005,09:26:04,"SSP\J. Vriens","Form submit",OK,"Print jobs - HP\_1220C"'

At 09:25:49, the forms available for the end user are loaded from the **UMRA Service**. At 09:25:53, the main form project is loaded. As part of the form generation process, the initial project is executed. At 09:26:04, the user selects a documents and presses the **Pause** button. The form information is submitted to the **UMRA Service** and the script of the project is executed: the print job is paused. Finally, the cycle starts over again: the script of the form initialization project is executed and the generated form is returned to the **UMRA Forms** application.

#### Project extensions

- The example project can easily be extended to support similar functionality:
- Include documents of multiple printers in a single table
- Show multiple tables with printer documents for multiple printers
- Setup a wizard to select a printer first, and then show the printer document table of the selected printer
- Add a button to reset the printer spooler service
- Filter out documents from the table from specific users.

For additional information and other UMRA example projects, contact Tools4ever at [www.tools4ever.com](http://www.tools4ever.com)  
<http://www.tools4ever.com>.



---

### 3.5. Managing Windows computer services

Although primarily focusing on user accounts and associated resources, you can also manage services using UMRA. From all computers, including domain controllers and regular workstations, the services can be managed. In this document, a sample project is described for managing Windows services.



*Read the full PDF version of UMRA Managing Windows computer services*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/Umra-Service-Management.pdf>

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

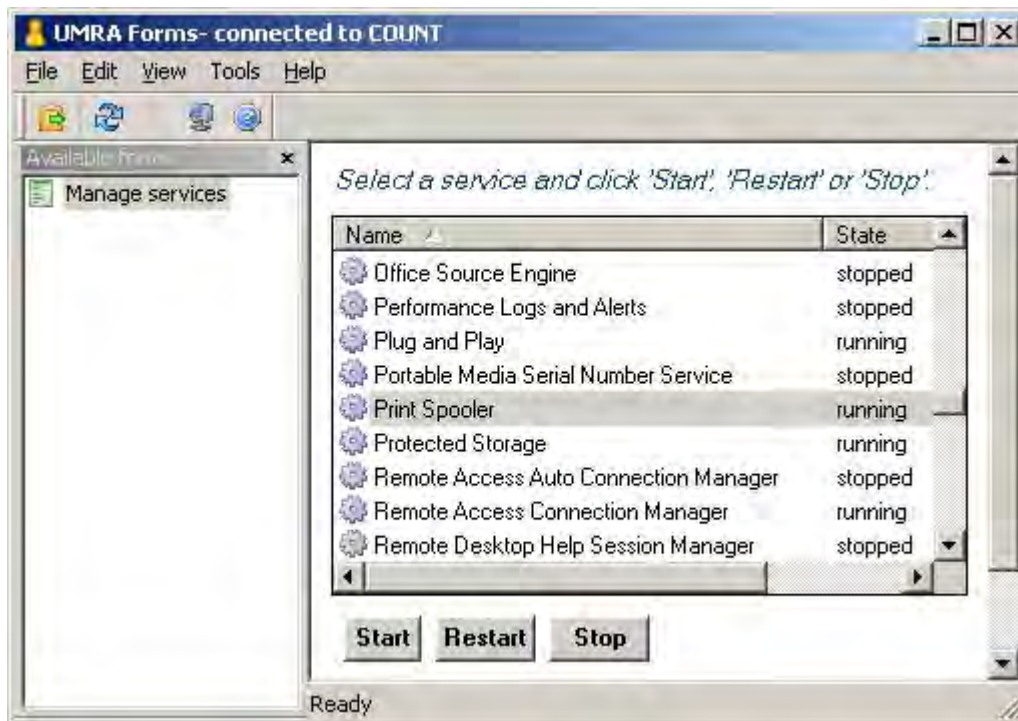
*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### 3.5.1. Project definition

In this example project, an implementation is described to (re)start and stop services that can be selected from a list.



In the form shown, the list presents an overview of all or a number of specific services from a specific computer. By selecting a service and clicking one of the buttons, the services can be managed.

1. The example project can easily be extended to:
2. Support multiple computers. For each computer, a form can be shown or the services of multiple computers are shown in a single form.
3. Only specific services are shown. This is especially useful in a helpdesk environment to allow employees to restart only specific services.
4. The number of commands can be limited. In this case, a user can only restart a not stop a service.

Note that the example project by default supports delegation and logging: Only specific users are granted access to run the form and all service management actions are logged.

### 3.5.2. Project structure

#### Form projects

The example scenario consists of 2 form projects:

1. **Manage Services** project: The main form project that holds the form and the script to manage the selected service. At initialization time, the **Manage Services** project accesses the other project.
2. **Collect Services** project: A very simple project that is used to collect the services information from a specific computer. The **Collect Services** project only contains a script and not a form. The script is used to collect the service information and store the services table in a variable.

Both form projects are available from the Tools4ever web-site. The projects are designed in such a way that only minimal changes are required to make the projects work in your environment.

### Principle of operation

As an initialization project, the script of the **Collect Services** project is executed when the form of the **Manage Services** project is executed. The script of the **Collect Services** projects collects the services information and stores all data in a single variable. The variable is shown as a table in the **Manage Services** project. When a service is selected from the list and one of the buttons is clicked, the script of the **Manage Services** project is executed. Next, the complete cycle starts over again, and again...

### 3.5.3. Step 1: Environment setup

#### Prerequisites

1. To run the project successfully you need to meet the following requirements:
2. You need to be logged on to the network with administrative privileges. During the implementation of the project, the UMRA Service is installed. The service can be installed on any computer but needs to have access to the computer with the services you need to manage.
3. The computer on which UMRA is installed must run one of the following operating systems: Windows XP, Windows 2000 (all versions) or Windows 2003 (all versions).

#### UMRA installation

To run the project, you need to install several UMRA modules. Once installed, you can run the product for 30 demo days. After this period, a valid license code is required. To start, download the UMRA software from [www.tools4ever.com](http://www.tools4ever.com) <http://www.tools4ever.com> and install at least the modules *UMRA Console* and *UMRA Forms*.

#### UMRA setup

Once installed, start the *UMRA Console* application: Select menu option **All programs, User Management Resource Administrator, UMRA Console**. Upon startup, the **User Management Resource Administrator Wizard** is started automatically. You can either run the wizard to become familiar with the product or move forward and start with the installation of the UMRA Service. To do so, **Cancel** out of the wizard and select **UMRA Service, Install or upgrade service**. Follow the instructions of the application.

It is advised to install the UMRA Service on a computer that is a member of the domain containing the user accounts and computer services you wish to manage. For test purposes, you can also install the **UMRA Service** on the same computer that runs the **UMRA Console** application.

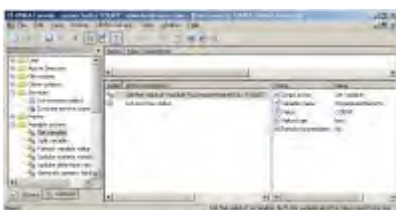
#### 3.5.4. Step 2: Form project - Collect Services

In this step, the auxiliary project **Collect Services** is created. As the name describes, the **Collect Services** project is used only to collect the information of a specific computer. The project stores the services information in a table variable that is used by the main project.

##### Starting the UMRA Console

1. Start the **UMRA Console** application and connect to the **UMRA Service**: Select **UMRA Service, Connect...** and connect to the computer on which the **UMRA Service** is installed.
2. To start creating the new form project, select **File, New....** Check button **Form project** and press **OK**. Enter the name of the project, **Collect Services** and press **OK**.
3. The form project window is initially empty. Only the **Window Help** sections are shown. To hide the **Window Help** sections, right click in one of the three window areas and deselect menu option **Show Window Help**.
4. This project is meant only to collect services information using a small script. Therefore, the project contains no form and the upper half of the window is not used to design a form.

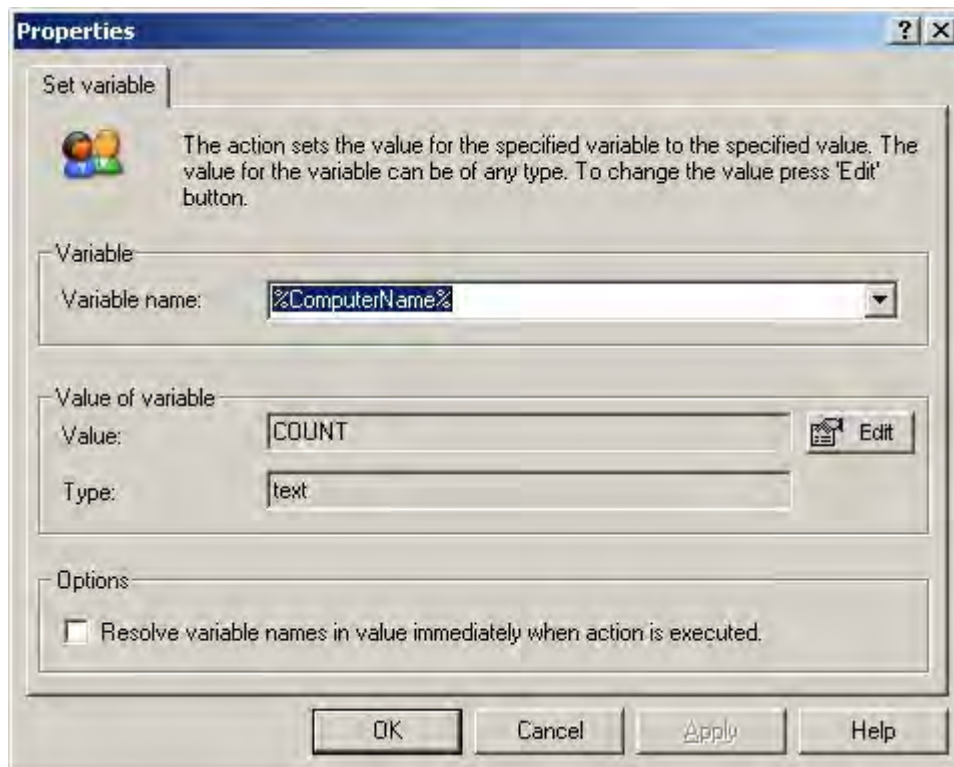
Next, we will setup the action that creates and initializes the variable that holds the name of the computer from which we want to manage the services. In the **Actions – Network** bar, activate the **Actions** window and expand the tree **Variable actions, Variable Operations**. Add the action **Set variable** to the script of project: drag- and drop the action to the lower left area of the window.



##### Setting up the Set variable action

1. Select the new action **Set the value of variable ...** in the lower left window of the project window (not in the **Actions** bar). In the lower right window, the properties of this action are now shown. Double click one of the properties (example: **Variable name** or select main menu action

Now setup the **Set variable** action as shown below:



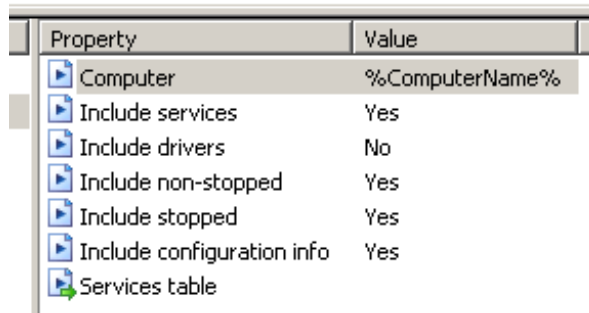
The action will create a variable with the name **%ComputerName%**. The variable will hold the name of the computer from which the services are managed. Press **Edit** to specify the name of the computer. In the example shown, computer **COUNT** is specified. Note: the variable **%ComputerName%** is used both in this project, and in the next project of the wizard: **Manage Services**. The variable is initialized only once.

#### Setting up the List services status action

1. Set up the action to collect the service status information. From the **actions** bar, select action **Services, List services status** and drag and drop the action to the script section (lower left) area of the project window. The script action is automatically selected and the properties section (lower right of project window) shows all the properties of this action. You need to setup the properties of this action one-by-one. By double-clicking each of the properties, you can specify the value of the selected property.
2. The most important properties of the **List services status** action are **Computer** and **Services table**. For **Computer**, specify the name of the variable created with previous action **Set variable**: **%ComputerName%**. The property **Services table** is by default configured as output variable **%ServicesTable%**. The green arrow shown with property **Service table** indicates that the value of the property is stored in an **output** variable. By double-clicking the property, you can setup



the property.



Property	Value
Computer	%ComputerName%
Include services	Yes
Include drivers	No
Include non-stopped	Yes
Include stopped	Yes
Include configuration info	Yes
Services table	

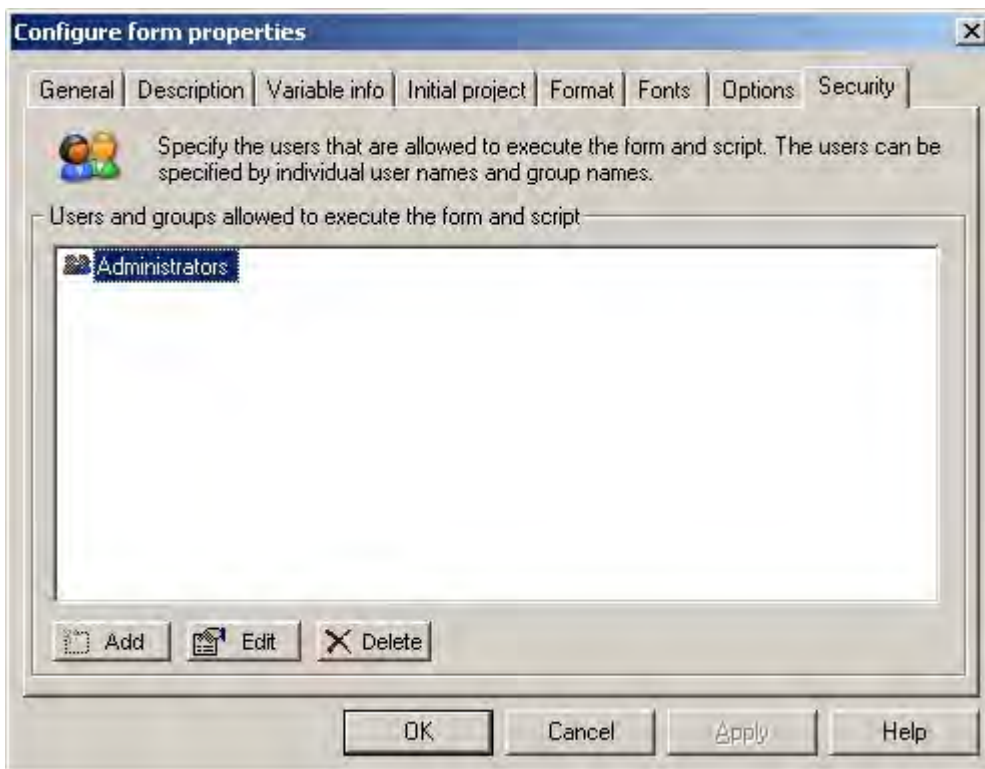
What happens when this action is executed: The UMRA software connects to the computer specified by **%ComputerName%** and collects the status of all of the services. The status information includes, the name of the service, the operational state of each service (running, stopped), type of service (automatic, manual, disabled) and so on. This information is stored as a table in variable **%ServicesTable%**. Note that the single variable will hold a table with multiple rows and columns. The variable is used in the other project.

### Setup project security

The project **Collect Services** is now almost complete. What remains to be done is to set up the security settings of the project (i.e. specify who is allowed to execute this form project).

1. Select the menu option **Actions, Form properties** and click the **Security** tab.
2. Press the **Add** button and enter the name of the user or group to setup the form project access rights. Press **OK** to finish this step.

3. Finally, save the project and close the project window.



### Summary

The project **Collect Services** is now ready for use and has the following characteristics:

- The project has no form, only a script
- The script sets a variable `%ComputerName%` to a specific value and collects the services information from the specified computer. The results are stored as a table in variable `%ServicesTable%`.
- The form project access rights are set up

### 3.5.5. Step 3: Form project - Manage Services

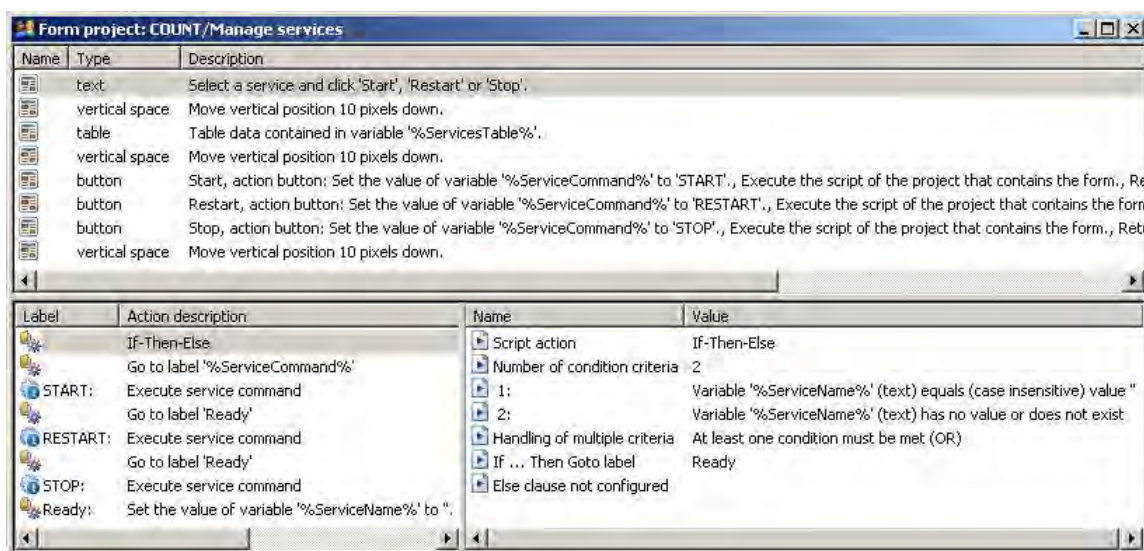
The **Manage Services** project is the main project of this example scenario. The project contains both a form and script. The form shows the services of the computer and buttons to manage the selected service:



The script of the project executes the **Start**, **Restart** or **Stop** action. When the script action is completed, the list with service status information is refreshed to reflect the new service status.

### Manage Services - Form, part 1

Start a new form project with the name **Manage Services**. See the previous section for more information on how to do this. To setup the form project, the form and the script must be designed. To setup the form, a number of form fields must be added to the form. For each form field, parameters must be specified.



To add a form field, right click in the upper form area of the project window and select menu option **Add form field....**

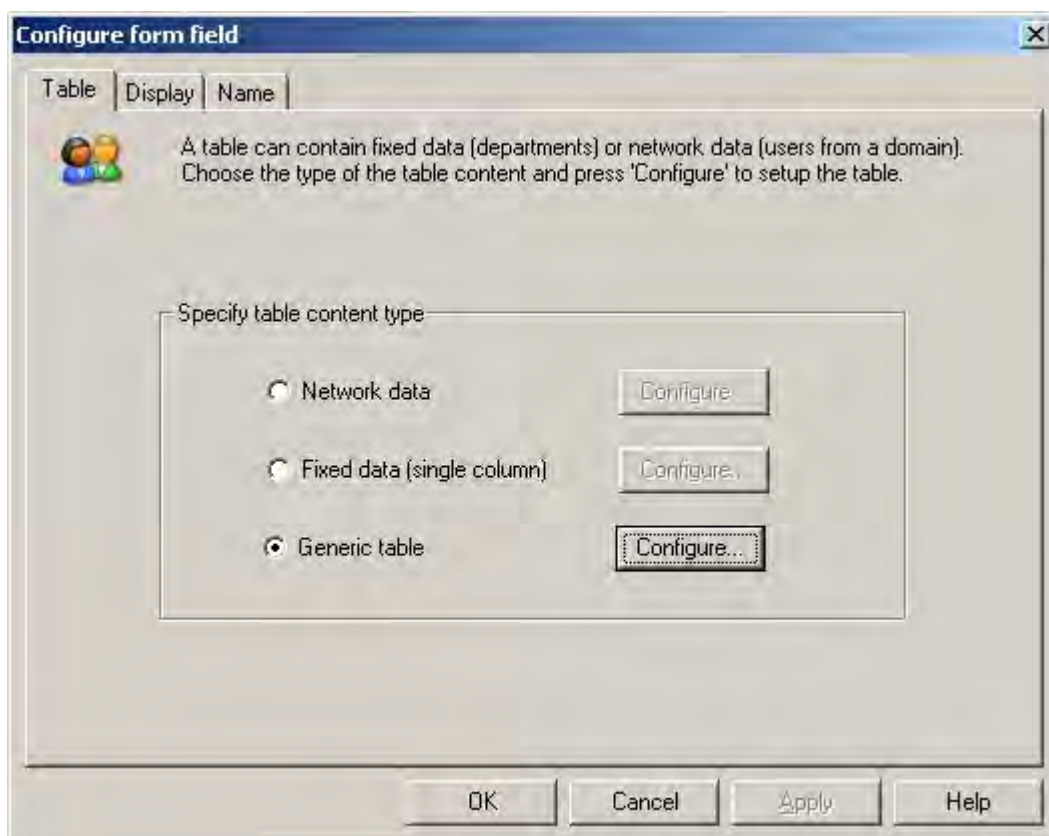
### Manage Services - Adding form fields

Add the following form fields:

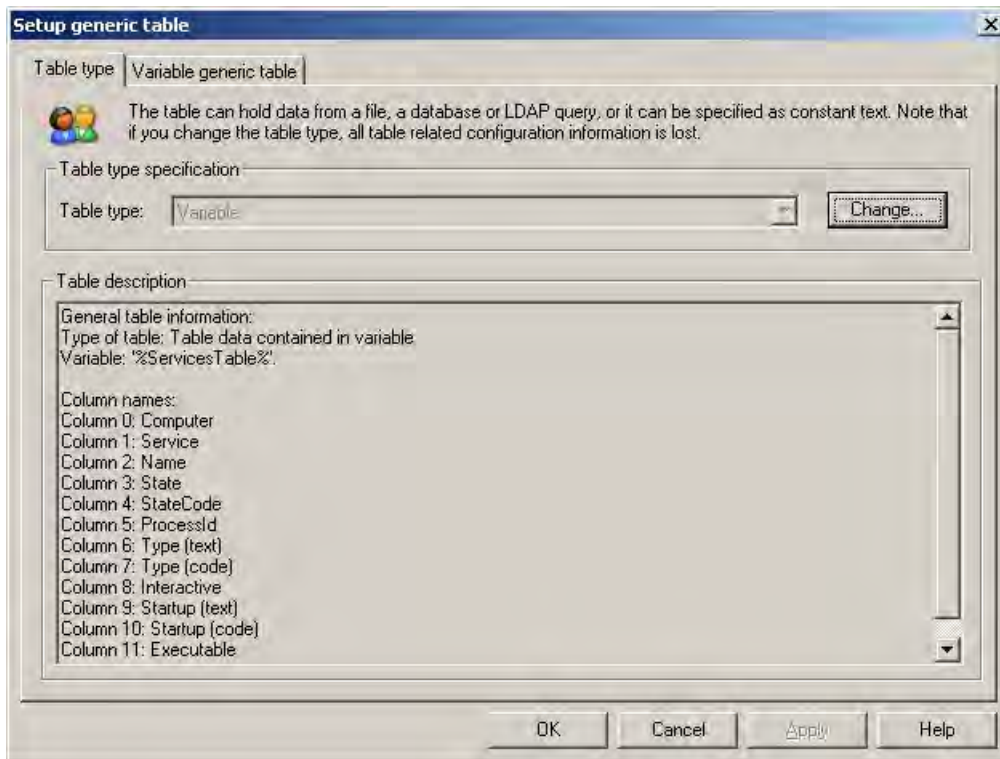
1. Static text field: The field shows the introduction text: *Select a service and click 'Start', 'Restart', or 'Stop'.*
2. Vertical space: Some open space (10 pixels) to outline the form.
3. Table: This is the table form field that lists the services. The table form field configuration is described in the next chapter.

### Manage Services - Form table

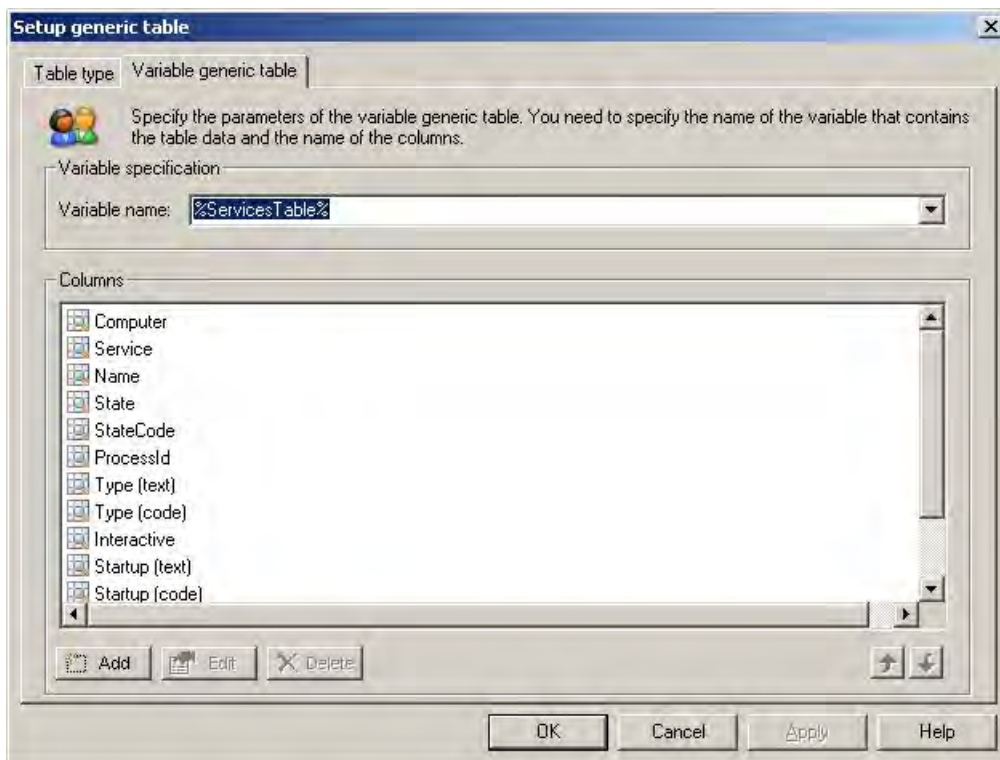
The form table lists the services and services status information. The configuration of the table is described in this chapter. The services information is obtained in project **Collect Services**. In that project, the services information is stored in a table variable. The variable is passed to the **Manage Services** project. The **generic table** type is able to show the table data of a variable. So select **Generic table** as shown in the following figure:



Press **Configure** to continue. Next, you need to select the type (source) of the generic table. Select **Variable** since the table data is obtained from a variable.



When selected, the configuration window **Variable generic table** can be selected. The window is used to specify the name of the variable and to define the columns of the table.

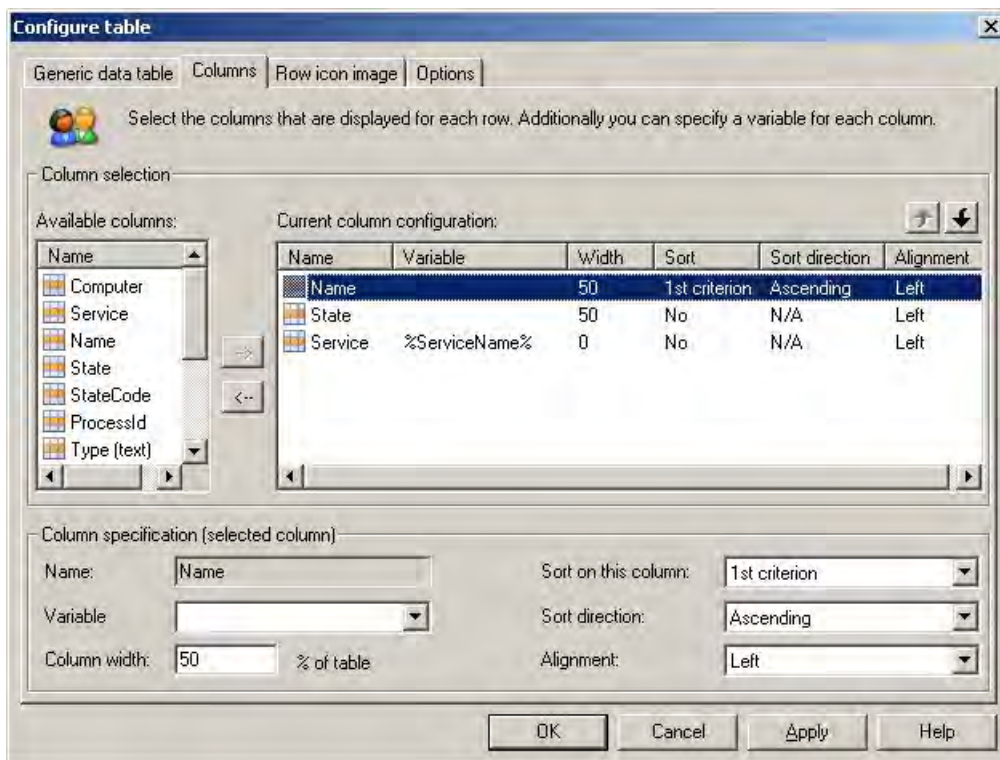


Specify the name of the variable: **%ServicesTable%**. The name must equal the name of the variable generated by the action **List services status** of project **Collect Services**.

Important: A table variable only holds the data of the table, not the names of columns. You therefore need to add the names of the columns that can be shown with the generic table. As described in the online help, the **List services status** generates a table with the following columns: Computer, Service, Name, State, StateCode, ProcessId, Type (text), Type (code), Interactive, Startup (text), Startup (code), Executable, Log on as. Add these columns one-by-one. The first part of the generic table configuration is now complete. Press **OK**.

You now need to setup the table columns that must be shown in the form. Click on the **Columns** tab.





This window is used to configure the columns that must shown in the form and to specify the variables that are passed to the UMRA Service when the end-user selects a service and presses a submit button.

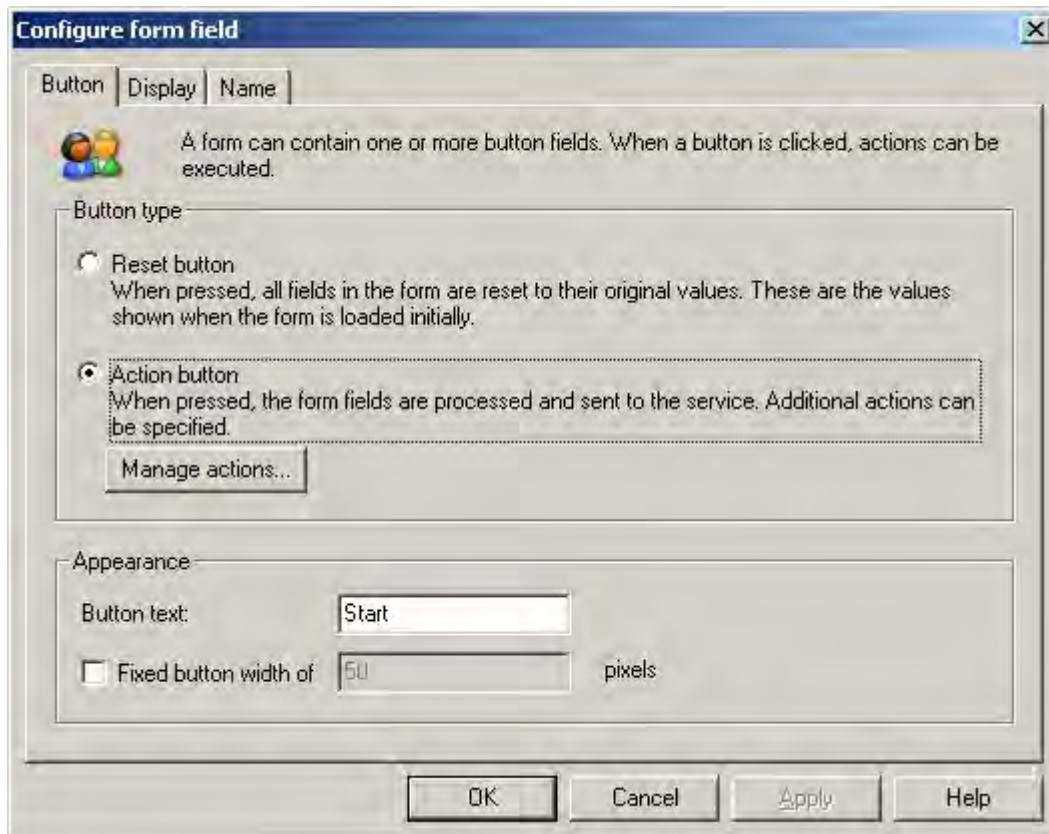
On the left side, the **Available columns** are shown. These columns correspond with the columns configured in the previous step. By using the add (->) and remove (<-) buttons you can setup a column configuration. In the example, the form will show a table with 3 columns. The 3<sup>rd</sup> column is not visible since it has a width of 0%. This column is included since it uniquely specifies the name of the service. When the user selects a service and presses a button, the value of this column is stored in variable **%ServiceName%**. This variable is passed to the UMRA Service and used for further processing.

Use the **Row icon image** and **Options** windows to specify additional table configuration settings. The configuration of the table is now complete.

To specify additional table display settings such as font and alignment, select the **Display** window of the **Configure form field** window.

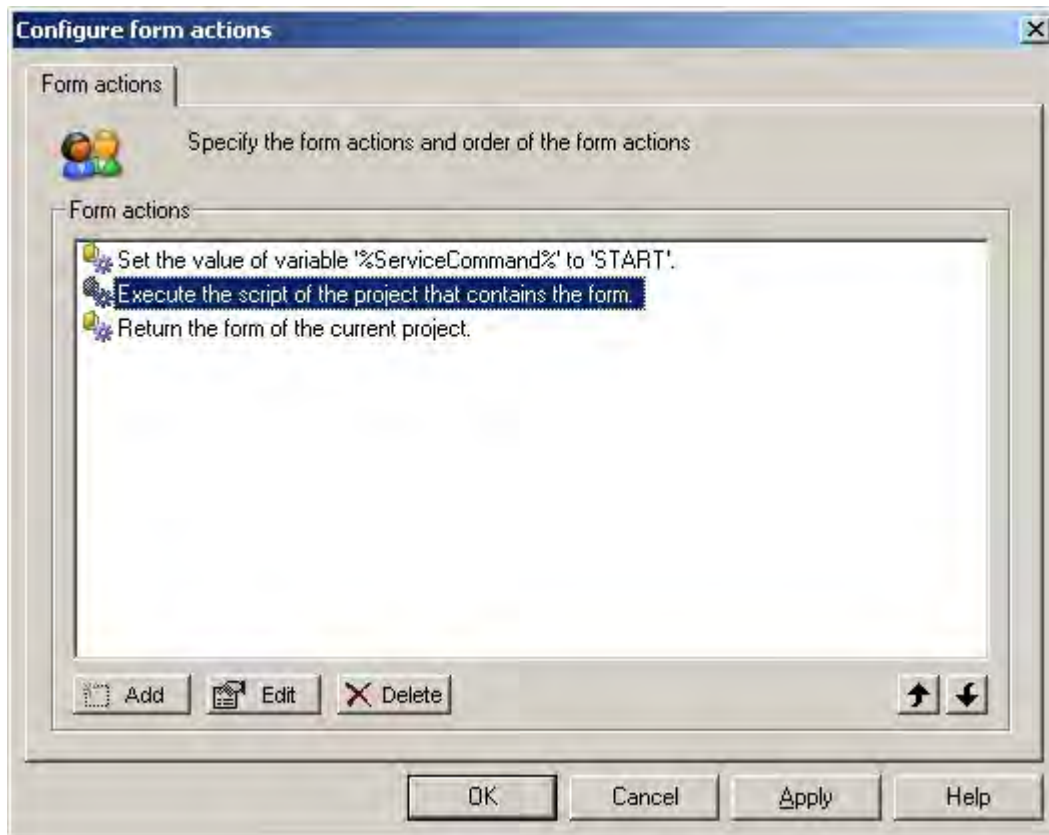
### Manage Services - Form buttons

In the example project, after the table form field, some vertical space is added. Next, the 3 submit buttons are configured. To setup the **Start** button, specify the **button type** as **Action button** and enter the text **Start** as **button text**.

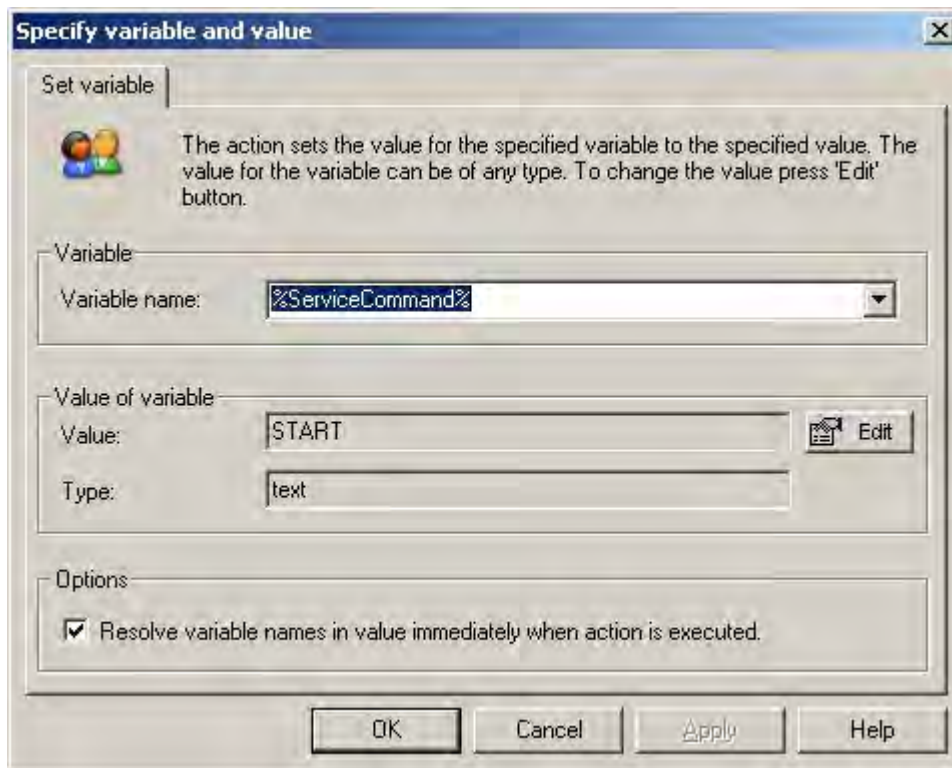




Press the **Manage actions** button to configure the actions that must be executed when the button is pressed in the form.

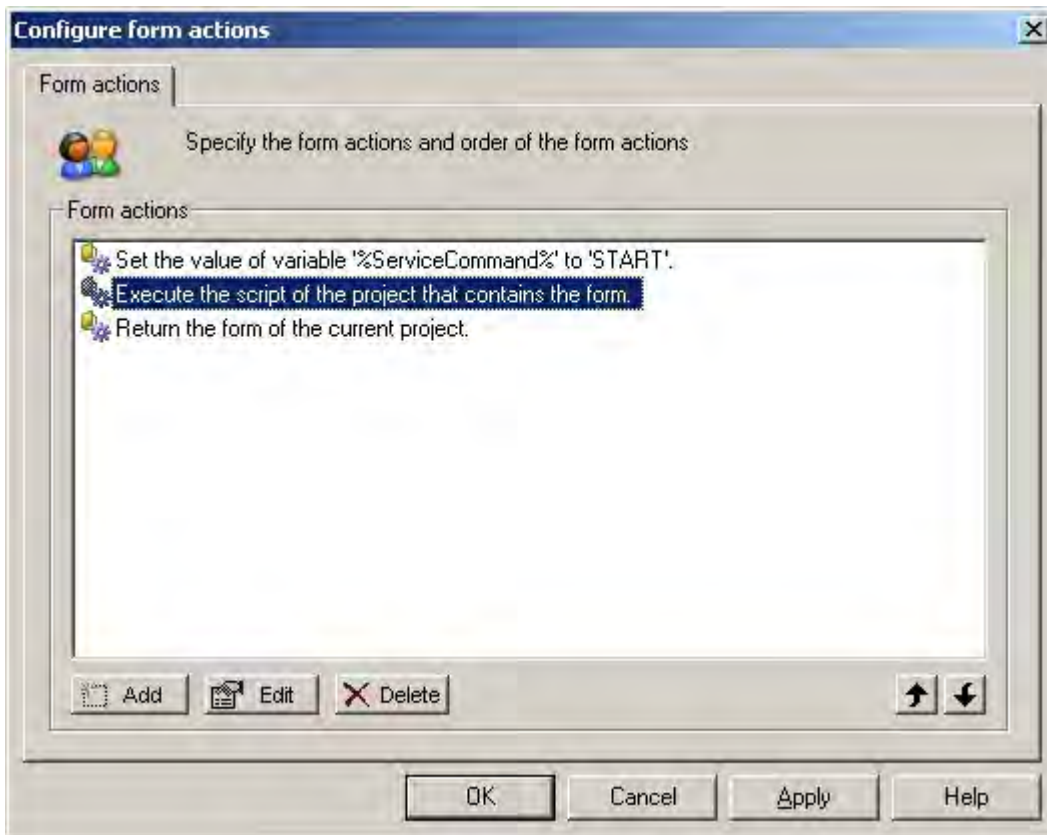


The script of the **Manage Services** project uses the variable **%ServiceCommand%** to jump to the appropriate label of the script. This variable is set as the first action of the script. For the start button, the variable is set to **START**.



So when the user presses the **Start** button, the value of the variable **%ServiceCommand%** is set to the text **START**. Similarly, the value of the variable **%ServiceCommand%** is set to **RESTART** and **STOP** for the other buttons.

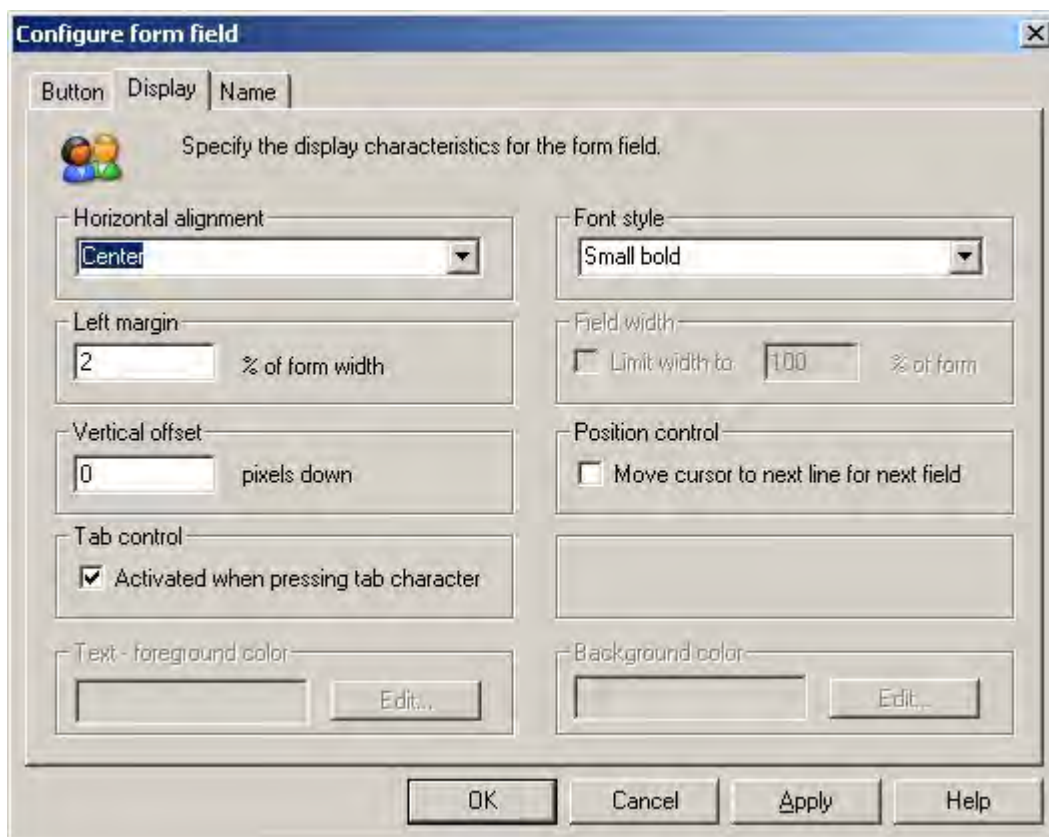
The next action instructs the UMRA Service to execute the script of the **Manage Services** project. This action has no parameters.



Finally, the last action executed when the button is pressed, returns the form of the project. This action needs no additional configuration settings.

The **Restart** and **Stop** buttons are identical to the **Start** button, except for the variable `%ServiceCommand%` as described earlier.

To position the buttons next to each other, configure the display settings for the **Start** and **Restart** buttons as follows:



Note the **left margin** of 2% to separate the buttons. The **Position control** setting **Move cursor to next line for next field** must be unchecked to position the buttons next to each other. For the last button, **Stop**, this option must be checked.

#### Manage Services - Script

When one of the submit buttons, **Start**, **Restart** or **Stop** is pressed, the script of the **Manage Services** project is executed as part of the submit button script action execution sequence.

The script uses the following variables:

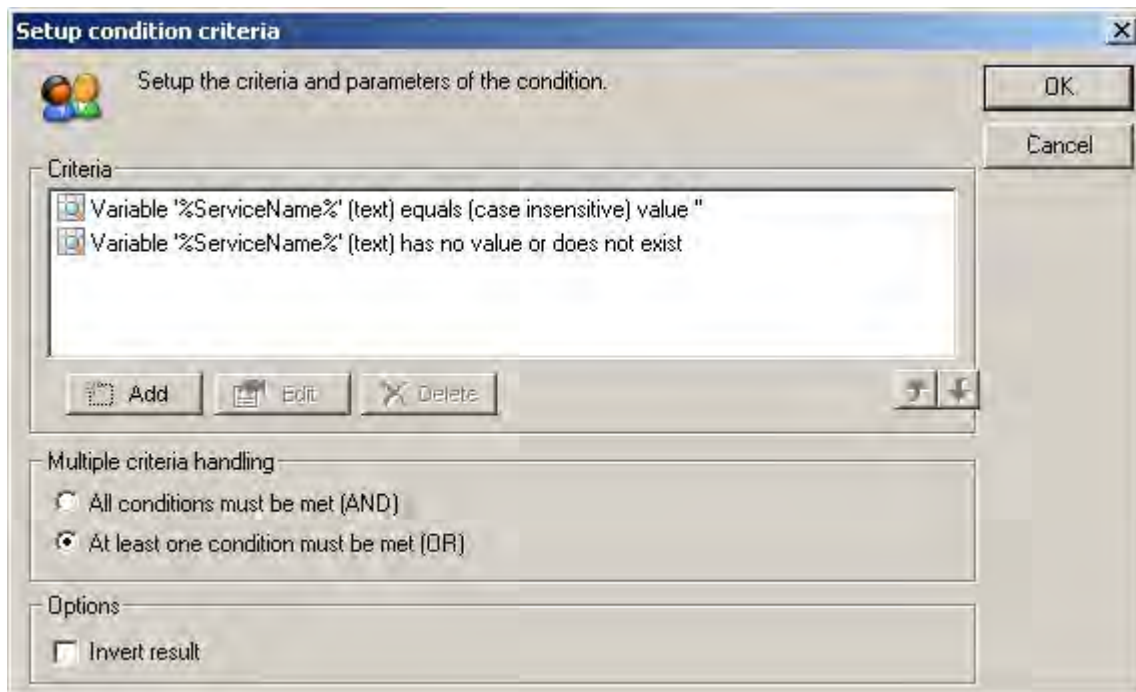
Variable	Description
%ServiceName%	The name of the service that is selected by the end-user in the form. When the user selects no service, the variable is empty. Otherwise, the value of the variable corresponds with the 3 <sup>rd</sup> column with zero width of the table shown in the form.

%ServiceCommand%	The value of the variable corresponds with the button pressed by the end-user. The value is set as an action when the button is pressed. For each button, the value is different. This button is executed, before the script of the project is executed.
%ComputerName%	The name of the computer that maintains the service. The variable is passed from the other project <b>Collect Services</b> . In a more realistic environment, the variable could be generated or selected in a window of a sequence of wizard windows.

The script first performs some input checking control and then jumps to the correct label and executes the requested action.



The action checks if the script input variable **%ServiceName%** is empty or does not exist. If this is the case, the end-user did not select a service from the list with services in the form, when one of the buttons was pressed.



When the variable **%ServiceName%** does not hold a valid value, the script execution proceeds with the action with label **Ready**.

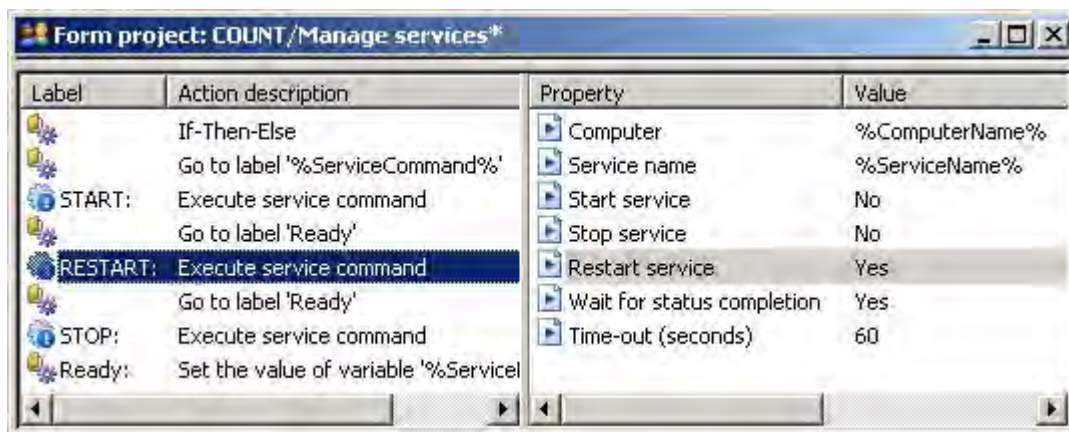
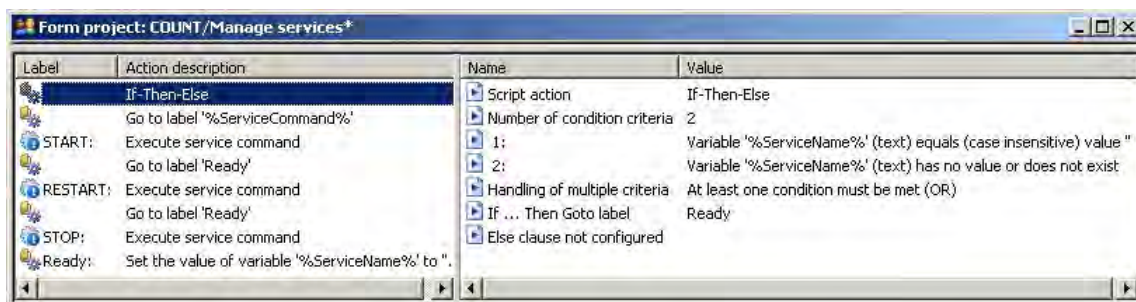
Next, the script jumps to the section that corresponds with the button pressed.





Note that the variable **%ServiceCommand%** is set as an action when one of the submit buttons is pressed. The value of the variable corresponds with the button: **START**, **RESTART** or **STOP**. In the script, exactly these values are used as labels. The **Go to label %ServiceCommand%** simply continues script action execution at one of the labels.

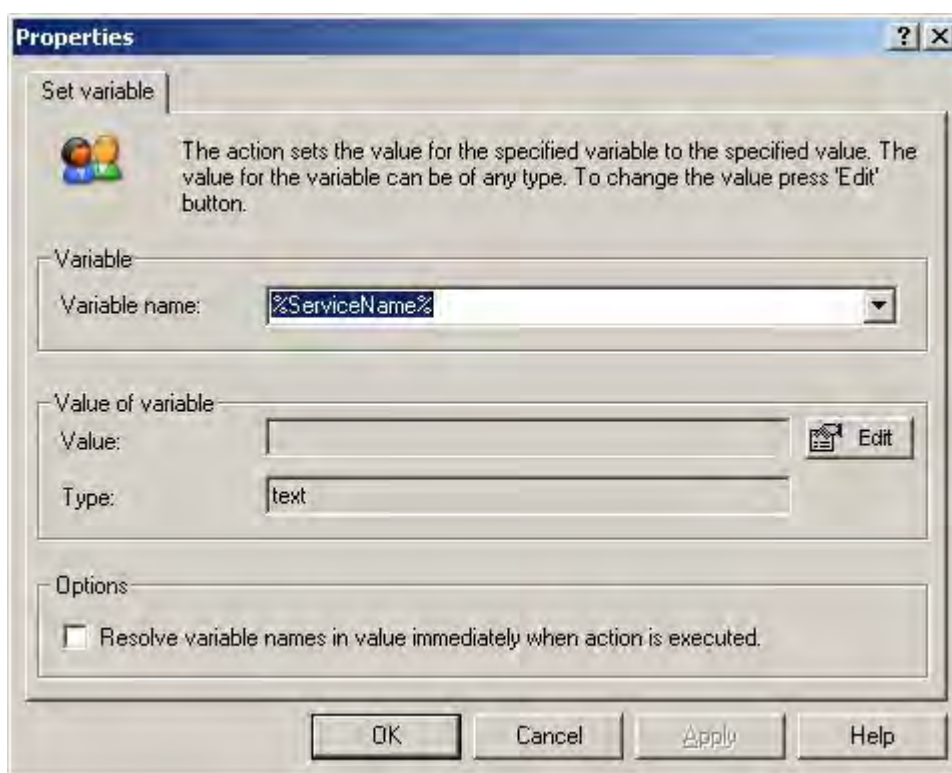
The **Execute service command** actions are simple to configure. For each of the actions, the **Computer** property is set to the variable **%ComputerName%** that is passed from project **Collect Services**.



The **%ServiceName%** variable is the result from the form. The **Start service**, **Stop service** and **Restart service** properties are configured according to the desired service action.

The **Execute service command** action is followed by a **Go to label 'Ready'** action to continue script execution at the right location.

Finally, the value of variable **%ServiceName%** is cleared to reset the input for the next form execution cycle.



Note that when the script is executed, the next action of the form submit button action sequence is executed: **Return the form of the current project**.

#### Manage Services - Link to project Collect Services

As described earlier, the form of the **Manage Services** project shows the services information collected in the script of project **Collect Services**. To achieve this behavior, the **Manage Services** project must be configured to execute the script of the project **Collect Services** *before the form of the Manage Services project is shown*.

Right click in the upper form section of the **Manage Services** project window and select **Form properties....** Select tab **Initial project** and specify project **Collect Services**. The script of the **Collect**



**Services** project will now be executed, each time the **form** of the **Manage Services** project is shown. The variables that are generated in the **Collect Services** project can be used in the form fields of the **Manage Services** project.

To finish the project, select the **Security** tab to setup the access rights for the **Manage Services** project.

### 3.5.6. Project execution

Now what happens when the user select the form **Manage Services** in the **UMRA Forms** application?

1. A request is sent to the from the **UMRA Forms** application to the **UMRA Service** to generate and return the **Manage Services** form.
2. The **UMRA Service** checks the access rights of the end-user and the **Manage Services** project is loaded by the **UMRA Service** and the form generation is initialized.
3. As part of the **Manage Services** form generation process, the project **Collect Services** is loaded and the script is executed. Resulting variables (**%ServicesTable%**) are stored and passed to the form generation process.
4. The form of the **Manage Service** project is generated. The table holds the data from the variable generated by the **Collect Services** project.
5. The form is returned to the **UMRA Forms** application and shown.
6. The user selects a service and presses one of the buttons.
7. The selected service is stored in variable **%ServiceName%** and send with information of the pressed button to the **UMRA Service**.
8. The **UMRA Service** checks the access rights of the end-user and the actions configured for the button are executed.
9. The form of the **Manage Services** project is generated and returned to the **UMRA Forms** application. As part of the form generation process, the script of the **Collect Services** project is executed.

The following section shows the **UMRA Service** log file information for a complete session:

12:19:47 09/21/2005 Form message: '09/21/2005,12:19:47,"SSP\J. Vriens","Forms list",OK,N/A,"1 projects found for user 'SSP\J. Vriens'.'"

12:19:49 09/21/2005 Executing form initialization project 'Collect Services'.

12:19:49 09/21/2005 Variable 1: %UmraFormSubmitAccount%=SSP\J. Vriens

12:19:49 09/21/2005 Getting services information from computer: 'COUNT'. Options: include services, exclude drivers, include non-stopped services and/or drivers, include stopped services and/or drivers, include configuration info.

12:19:49 09/21/2005 Form message: '09/21/2005,12:19:49,"SSP\J. Vriens","Form load",OK,"Manage services",'

12:19:54 09/21/2005 Variable 1: %ServiceName%=W3SVC

12:19:54 09/21/2005 Variable 2: %UmraFormSubmitAccount%=SSP\J. Vriens

12:19:54 09/21/2005 Variable 3: %ComputerName%=COUNT

12:19:54 09/21/2005 Variable 4: %ServicesTable%=Table with 96 rows

12:19:54 09/21/2005 Variable 5: %NowDay%=21

12:19:54 09/21/2005 Variable 6: %NowMonth%=09

12:19:54 09/21/2005 Variable 7: %NowYear%=2005

12:19:54 09/21/2005 Variable 8: %NowHour%=12

12:19:54 09/21/2005 Variable 9: %NowMinute%=19

12:19:54 09/21/2005 Variable 10: %NowSecond%=54

12:19:54 09/21/2005 Variable 11: %ServiceCommand%=RESTART

12:19:54 09/21/2005 If-Then-Else condition [Variable '%ServiceName%' (text) equals (case insensitive) value " OR Variable '%ServiceName%' (text) has no value or does not exist] result is FALSE, continue script execution with next action.

12:19:54 09/21/2005 Jump to script action with label 'RESTART'.

12:19:54 09/21/2005 Executing command for service 'W3SVC' on computer 'COUNT': Restart service.

12:19:54 09/21/2005 Waiting 60 seconds for status completion.

12:19:56 09/21/2005 Service successfully 'restarted (stopped)'.

12:19:57 09/21/2005 Service successfully 'restarted (started)'.

12:19:57 09/21/2005 Jump to script action with label 'Ready'.

12:19:57 09/21/2005 Executing form initialization project 'Collect Services'.

12:19:57 09/21/2005 Variable 1: %ServiceName%=

12:19:57 09/21/2005 Variable 2: %UmraFormSubmitAccount%=SSP\J. Vriens

12:19:57 09/21/2005 Variable 3: %ComputerName%=COUNT

12:19:57 09/21/2005 Variable 4: %ServicesTable%=Table with 96 rows

12:19:57 09/21/2005 Variable 5: %NowDay%=21

12:19:57 09/21/2005 Variable 6: %NowMonth%=09

12:19:57 09/21/2005 Variable 7: %NowYear%=2005

12:19:57 09/21/2005 Variable 8: %NowHour%=12

12:19:57 09/21/2005 Variable 9: %NowMinute%=19

12:19:57 09/21/2005 Variable 10: %NowSecond%=54

12:19:57 09/21/2005 Variable 11: %ServiceCommand%=RESTART

12:19:57 09/21/2005 Getting services information from computer: 'COUNT'. Options: include services, exclude drivers, include non-stopped services and/or drivers, include stopped services and/or drivers, include configuration info.

12:19:57 09/21/2005 Form message: '09/21/2005,12:19:54,"SSP\J. Vriens","Form submit",OK,"Manage services"'

First, at 12:19:47, the forms are loaded into the UMRA Forms application. Then, starting at 12:19:49 the form of project **Manage Services** is generated. This includes the execution of project **Collect Services**. At 12:19:54 the web-service is selected from the list and the **Restart** submit button is pressed. The script of the **Manage Services** project is executed. At 12:19:54 the service is requested to stop. 2 seconds later, at 12:19:56, the service is stopped and started at 12:19:57. Next, the script of project **Collect Services** is executed as part of the form generation process of project **Manage Service** project. At 12:19:57 the form is returned to the **UMRA Forms** client application and the process is complete.

### 3.5.7. Contacts

More information can be found at the following locations:

<http://www.tools4ever.com>

<http://forum.tools4ever.com>



---

### 3.6. Managing LDAP directory services using UMRA

Although primarily focusing on Microsoft Active Directory, User Management Resource Administrator (UMRA) can also manage any other directory service, as long as the directory service supports the **Lightweight Directory Access Protocol (LDAP)**.

Examples of the directory services that can be managed with UMRA include Novell eDirectory, Linux OpenLDAP and Microsoft's Active Directory.



*Read the full PDF version of UMRA Managing LDAP directory services*  
<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/Managing-Ldap-Directory-Services.pdf>

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### 3.6.1. Introduction

Although primarily focusing on Microsoft Active Directory, User Management Resource Administrator (UMRA) can also manage any other directory service, as long as the directory service supports the **Lightweight Directory Access Protocol (LDAP)**.

Examples of the directory services that can be managed with UMRA include Novell eDirectory, Linux OpenLDAP and Microsoft's Active Directory.

#### Main functions

The main reasons to use the LDAP functions of UMRA deal with the integration of networks with hybrid directory services. The UMRA LDAP functions include:

- Create user accounts and setup all attributes
- Manage group memberships
- Reset user account passwords
- Delete user accounts
- Manage all other directory service objects

With the UMRA LDAP functions, multiple directory services can be updated by executing a single task.

Example: When a form (of UMRA Forms and Delegation) is submitted, a user account can be created in Microsoft's Active Directory and Novell eDirectory in a single task.

#### Deployment scenarios

The UMRA LDAP functions are most often deployed for the following tasks:

- Synchronization of Active Directory updates with other directory services (Novell eDirectory, LINUX OpenLDAP);
- Synchronization of database system updates with (multiple) directory services;
- Helpdesk task delegation to manage the user account life cycle process, e.g. create user accounts, reset passwords etc. for hybrid directory services networks.

#### Secure LDAP (SSL) support

The LDAP functions of UMRA support both secure and non-secure LDAP implementations. Secure LDAP is implemented using SSL. The SASL authentication methods are not supported in UMRA.

### 3.6.2. Concept

#### **Directory Service Infrastructure**

The UMRA LDAP functions are typically used in a Microsoft Active Directory network environment with some other directory service that co-exists in the same network infrastructure. The other directory service is for instance Novell eDirectory or an OpenLDAP implementation on Linux. As long as the directory service supports LDAP, the directory service can be managed with UMRA.

#### **LDAP Server and LDAP Client**

The computer that runs the directory service and supports LDAP is referred to as the LDAP Server. The software that connects to the LDAP Server is referred to as the LDAP Client. According to these conventions, the UMRA software always acts as the LDAP Client and the contacted directory service system is the LDAP Server.



### Helpdesk scenario

In a helpdesk environment, the UMRA Forms client runs on a helpdesk computer. When a form is submitted by a helpdesk employee, the form and form input data is sent to the UMRA Service. The UMRA Service executes the script associated with the form. In a hybrid directory service environment, the script contains UMRA LDAP script actions to manage the LDAP directory service.

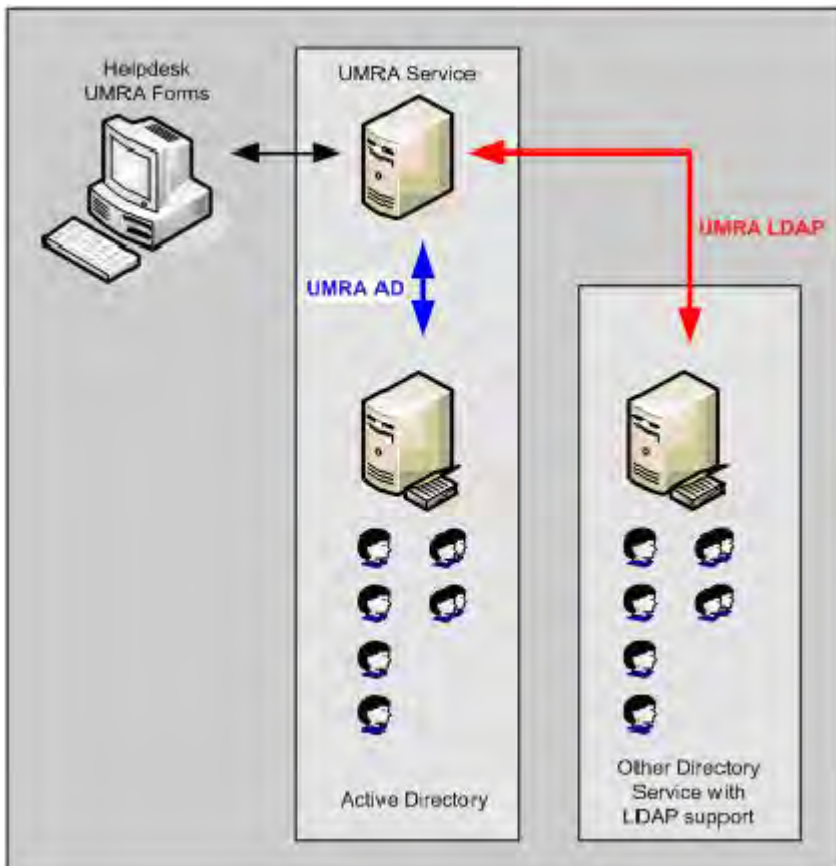


Figure 1: Network with helpdesk running UMRA Forms in a hybrid directory services network.

### Security

The LDAP protocol supports a large variety of features for security and authentication. With UMRA, 2 options are available:

- **Not secure:** All communication with the LDAP Server and the UMRA software is not encrypted. Authentication is accomplished using an account name and a password that is sent as clear text. Although simple to implement, this option is not recommended because of security reasons. The option can be used for testing purposes.

**Secure with SSL:** All communication between the LDAP client, e.g. the UMRA software and the LDAP Server is encrypted using the SSL standard. This option is recommended and secure. All data is encrypted.

To implement this option, SSL certificates need to be installed on both the LDAP Client and Server. The methods how to do this, largely depends on the implementation of the operating system and directory service. For Microsoft Active Directory, Novell eDirectory and Linux OpenLDAP the exact implementations are described in this document. For other systems, a similar approach must be used.

---

### 3.6.3. UMRA LDAP script actions

#### Script actions overview

In UMRA, a number of script actions are available to manage an LDAP directory service. The script actions cover the most important LDAP functions to update a directory service and execute a query in the directory service. All LDAP calls are executed synchronously.

#### Script actions

UMRA LDAP action	Description
<i>Setup LDAP session</i> on page 5	Initialize a secure or not secure LDAP session with the LDAP Server. The session parameters are stored in a variable that is used in subsequent UMRA LDAP actions.
<i>Load LDAP modification data</i> on page 7	Setup a data structure that is used to add and edit directory service items. The resulting LDAP modification data is stored in a variable that is referenced in subsequent script actions. The action is always used in combination with the Add and Edit UMRA LDAP actions.
<i>Add directory service object (LDAP)</i> on page 10	Add an item to the LDAP directory service. The data added is setup with action <b>Load LDAP modification data</b> .
<i>Modify directory service object (LDAP)</i> on page 10	Update an existing item in the directory service. The data used to modify the existing directory service item is setup with action <b>Load LDAP modification data</b> .
<i>Delete directory service object (LDAP)</i> on page 11	Delete an item from the directory service.
<i>Search directory service (LDAP)</i> on page 11	Execute a search action in the LDAP directory service. The results are returned in a generic table variable that can be used in subsequent script actions and forms.

Table 1: Overview of UMRA LDAP script actions

#### Script action: Setup LDAP session

For each action to update or search the LDAP directory service, a session must be initialized first. The session is most often initialized in the

beginning of the script and then used in all subsequent LDAP actions. The session is automatically released by the UMRA software when the script is finished.

Property	Description
LDAP server	The name of the host running the LDAP server. The name must be specified using the TCP/IP address or DNS name. Optionally, the name can be followed by a colon (:) and port number.
LDAP port	Optional: The TCP port number of the LDAP server to which to connect. The property is ignored if the specified 'LDAP server' includes a port number. If not specified, the default port is used. For not secure LDAP, the default LDAP port is 389, for secure LDAP (SSL), the default port is 636.
SSL encryption flag	If set to 'Yes', the session uses SSL encryption to communicate. In this case, on both the LDAP client and server side, appropriate SSL certificates need to be installed. If set to 'No', the action establishes a plain TCP connection and uses clear text (no encryption). Several topics in this document describe how to setup secure LDAP.
User name	The name of the user to connect to the LDAP server. If not specified, no user is authenticated, and no other LDAP actions can be executed. The format and exact name depends on the directory service.
User password	The password of the user specified with property <b>User name</b> . Note that the password is stored with encryption.
LDAP session	A data structure representing the resulting LDAP session. This property is an <b>output only</b> property and is generated automatically. The data is stored in a variable. (Default name: <b>%LdapSession%</b> ) This property is used in other script actions.

Table 2: Properties of action Setup LDAP session

The LDAP session variable can be passed to other scripts that are executed within the context of the outer script. When the outer script ends, the LDAP session is released.

**Script action: Load LDAP modification data**

When a directory service is updated to create a new item or update an existing item, the operation is always specified by the one or more attributes, the attribute value(s) and the type of attribute value modification: add, delete or replace.

To support this mechanism, the script action is **Load LDAP modification data** is used. All attributes, attribute values and value modification types are specified with this action. The result is stored in a variable that holds all the attribute information. The variable is then used in the action to:

- Create the item with action **Add directory service object (LDAP)** on page 10
- or
- Update the item with action **Modify directory service object (LDAP)** on page 10.

The action **Load LDAP modification data** does not communicate with the LDAP Server, that is, no session variable is required.

The **LDAP modification data** window is used to specify the LDAP modification data.

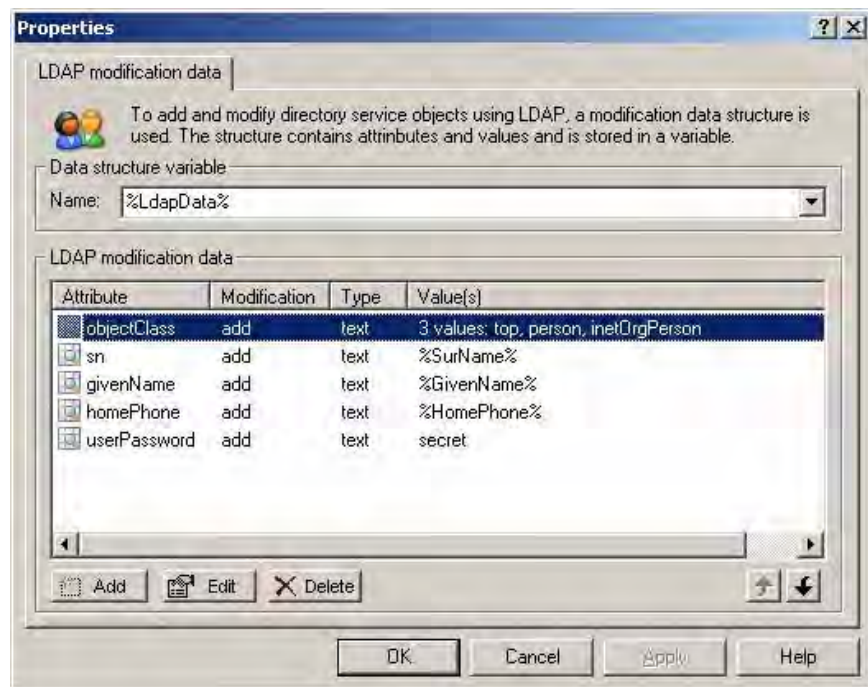
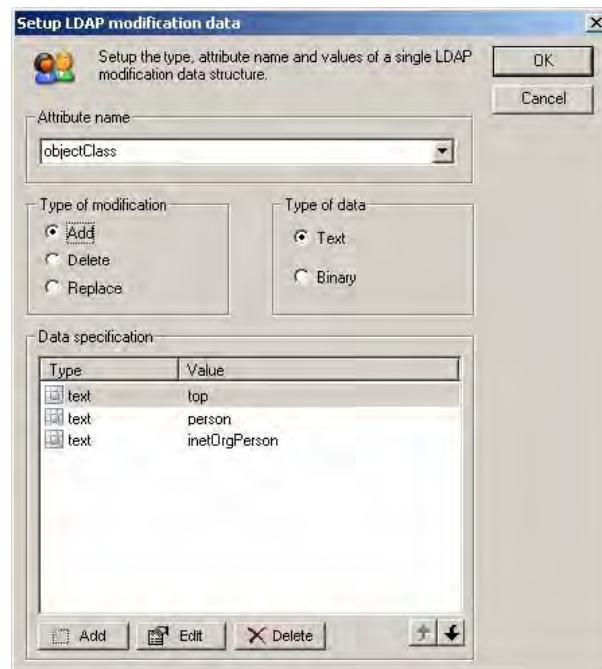


Figure 2: Specification of the LDAP modification data

In the example shown, the data is stored in variable **%LdapData%**. The data holds the modification values for 5 attributes: **objectClass**, **sn**, **givenName**, **homePhone** and **userPassword**. The names of the attributes are specified using their LDAP names as specified in the schema of the directory service. The values for each attribute can be specified using variables. Each attribute can have one or more values.

### Directory service schema

To specify the values of an attribute, the **Setup LDAP modification data** window is used.



*Figure 3: Multiple value specification for a single attribute*

For the attribute you need to specify the following:

1. **Type of modification:** Either **Add**, **Delete** or **Replace**, depending on the required type of modification.  
**Add:** add the specified values to the attribute. Existing attributes values are not removed. If the attribute already contains one of the specified values, an error occurs.  
**Delete:** delete the specified value from the attribute. If the specified value is not a value of the attribute, an error occurs.  
**Replace:** delete all of the existing attribute values and add the specified values to the attribute.
2. **Data specification:** The values of the attribute. The values can be specified as fixed values or variables.
3. **Type of data:** The type of the data, either text or binary. Almost all attribute values, including text, numbers, Boolean flags, date and time values can be specified using text.

**Script action: Add directory service object (LDAP)**

The action is used to add a new item to the directory service. The item is identified by its name that must be unique. All other parameters of the item are specified by its attributes. Before this action, the following actions must have been executed:

- **Setup LDAP session:** The session (data) is stored in a variable that is used in the action.
- **Load LDAP modification data:** Initialize the attributes and attribute values for the new directory service item.

The script action has the following properties:

Property	Description
LDAP session	A data structure representing a session with the LDAP server. The property is initialized with action <b>Setup LDAP session</b> and passed to this action using a variable. The default variable name is <b>%LdapSession%</b> .
New object name	The name of the new directory service object. The name must be specified as a distinguished name. Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com.
Object data	All attributes and values to add the object. The property must be specified as a variable name. This variable is generated by action <b>Setup LDAP modification data</b> .

*Table 3: Properties of action Add directory service object (LDAP).*

**Script action: Modify directory service object (LDAP)**

The action is used to update one or more attributes of an existing directory service item. The item is identified by its name (Object name) that is specified as a distinguished name. Before this action, the following actions must have been executed:

- **Setup LDAP session** on page 5: The session (data) is stored in a variable that is used in this action.
- **Load LDAP modification data** on page 7: Initialize the attributes and attribute values for the new directory service item.

The script action has the following properties:



Property	Description
LDAP session	A data structure representing a session with the LDAP server. The property is initialized with action Setup LDAP session and passed to this action using a variable. The default variable name is %LdapSession%.
Object name	The distinguished name of the object to modify. Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com.
Object data	All attributes and values to add the object. The property must be specified as a variable name. This variable is generated by action Setup LDAP modification data.

*Table 4: Properties of action Modify directory service object (LDAP).*

#### **Script action: Delete directory service object (LDAP)**

The action is used to delete an existing directory service item. Before this action is executed, an LDAP session must have been initialized with action **Setup LDAP session** on page 5.

The script action has the following properties:

Property	Description
LDAP session	A data structure representing a session with the LDAP server. The property is initialized with action <b>Setup LDAP session</b> and passed to this action using a variable. The default variable name is %LdapSession%.
Object to delete	The distinguished name of the object to be deleted. <b>Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com.</b>

*Table 5: Properties of action Delete directory service object (LDAP).*

If the item to delete does not exist, an error occurs.

#### **Script action: Search directory service (LDAP)**

This action is the general action used to search in the directory service.

The action is used for multiple purposes:

- To obtain a table of directory service item attribute values.  
Example: the name and description of the user accounts in a certain organization.

- To check the existence of an item.

The result of the search is always represented as a table that is stored in a single variable. In UMRA, the table can be used in 2 manners:

1. In forms, to show the contents of the resulting table to the end-user. The end-user can select one or more entries from the table.
2. In script actions, to evaluate the contents of the table and execute other script actions;

The search action is specified by a number of parameters using two windows.

The screenshot shows a 'Properties' dialog box with two tabs: 'LDAP search' and 'Attributes'. The 'LDAP search' tab is active. It contains a description: 'Perform a search on a LDAP server. On input, an LDAP session must exist. The result of the LDAP search is stored in a table.' Below this, there are two dropdown menus under 'Variables': 'Session:' set to '%LdapSession%' and 'Result:' set to '%UserTable%'. The 'Search specification' section has a 'Base (DN):' field with the value 'ou=UserAccounts,o=Tools4ever' and a 'Filter:' dropdown menu with the value '&{objectClass=user}(cn=H\*)'. The 'Scope' section has three radio buttons: 'Base only', 'One level', and 'Subtree', with 'Subtree' selected. The 'Options' section has two checkboxes: 'Time out interval' (unchecked) and 'Size limit' (checked). The 'Time out interval' has a value of '300' and the unit 'seconds'. The 'Size limit' has a value of '25000' and the unit 'entries'. At the bottom are buttons for 'OK', 'Cancel', 'Apply', and 'Help'.

Figure 4: LDAP search specification

The **LDAP Search** window is used to specify the search:

**Session:** The variable representing the **LDAP Session** that is initialized with action **Setup LDAP session**.

**Result:** The name of the variable that is used to store the result of the search. The search result is always stored as a table. The variable does not need to exist when the action is executed. If it does exist, the old value is overwritten.

**Base (DN):** The distinguished name of the directory service tree where the search should start. The search is executed at the specified base, and optionally in the immediate or all subtrees of the directory service.

**Filter:** The specification of the filter to perform the search. The standard search specification according to RFC2254 can be used to execute the search.

**Scope - Base only:** Limit the search to the specified base only. The maximum number of matching directory service items is 1.

**Scope - One level:** The search is performed in all entries of the first level below the base entry, excluding the base entry.

**Scope - Subtree:** The search is performed in the base entry and all levels below the base entry.

**Time out interval:** When enabled, the specified value is the time-out value of the LDAP search and the operation time. If disabled, no time-out value is used.

**Size limit:** When enabled, the maximum number of matching values is limited to the specified value. When disabled, the maximum number of items is not limited.

In the **Attributes** window, the attributes that must be returned for each matching directory service item are specified. The attributes are specified using the LDAP name as specified in the schema definition of the directory service.



*Figure 5: LDAP search attribute specification.*

#### **Distinguished name**

The result of the search is a table. In the table, the rows correspond with matching directory service items. Each column corresponds with an attribute. The distinguished name is by default stored in the last column of the table. So the example shown in the above figure, results in a table with 4 columns. The distinguished name is normally used to identify a directory service item.

Note: The names of the columns are not stored as part of the table data. If the variable is in a form to show the table data, the column names need to be specified as part of the table form field specification.

---

#### 3.6.4. Directory Service tasks

The way in which the LDAP script actions are used for the different directory service tasks, is described in this section.

##### Creating a directory service item

To create an item in the directory service (a user account, for instance) the following script actions are used:

1. **Setup LDAP session** on page 5: Connect to the LDAP Server and authenticate the directory service user account that is used to perform the update. Depending on the configuration, a secure session can be initialized;
2. **Load LDAP modification data** on page 7: Initialize all the attributes and attributes values that are required to create the directory service item. The exact attributes and values used vary for each directory service and are determined by the directory service schema;
3. **Add directory service object (LDAP)** on page 10: Add the item to the directory service using the LDAP modification data prepared in the previous step.

The attributes and attribute values vary for each directory service. This document contains examples that show how to create a user account in Novell eDirectory, LINUX OpenLDAP and Microsoft Active Directory.

##### Updating directory service item attributes

With an LDAP directory service, directory service management always deals with directory service items, the attributes of these items and the values of the attributes. For instance, to make a user account a member of a group in Novell eDirectory, 2 attributes must be updated for both the user account and group directory service item.

When updating directory service item attributes, the item must already exist in the directory service. The following operations can be performed:

- Add a value to an existing attribute;
- Delete a value from an existing attribute;
- Update the value(s) of an existing attribute;

- Add an attribute and value to an directory service item
- Delete an attribute and all values from a directory service item.

The following section lists the general procedure to update the directory service attributes:

1. **Setup LDAP session** on page 5: Connect to the LDAP Server and authenticate the directory service user account that is used to perform the update. Depending on the configuration, a secure session can be initialized;
2. **Load LDAP modification data** on page 7: Initialize all the attributes and attributes values that are required to update the directory service item. The exact attributes and values used vary for each directory service and are determined by the directory service schema;
3. **Modify directory service object (LDAP)** on page 10: Modify one or more of the attributes of the existing directory service item using the LDAP modification data prepared in the previous step.

This document contains multiple examples to update attributes for Novell eDirectory, LINUX OpenLDAP and Microsoft Active Directory items.

#### **Deleting a directory service item**

The following section lists the general procedure to delete a directory service item:

1. **Setup LDAP session** on page 5: Connect to the LDAP Server and authenticate the directory service user account that is used to perform the delete service item operation. Depending on the configuration, a secure session can be initialized;
2. **Delete directory service object (LDAP)** on page 11: Delete the item from the directory service.

This document contains several examples to delete directory service items.

**Searching a directory service (LDAP)**

UMRA supports the LDAP search specification RFC2254 to search in a directory service. Example: to find all users of which the common name (cn) starts with H, the following filter is used on Novell eDirectory:

```
(&(objectClass=user) (cn=H*))
```

Any filter can be used to return any collection of attribute values for the matching directory service items.

The following section summarizes the general procedure to search in the directory service:

1. **Setup LDAP session** on page 5: Connect to the LDAP Server and authenticate the directory service user account that is used to perform the search operation;
2. **Search directory service (LDAP)** on page 11: Perform a search operation in the directory service. The result of the search operation is a table that is stored in a variable.





---

### 3.6.5. Novell eDirectory

This section describes how user accounts in Novell eDirectory can be managed with User Management Resource Administrator (UMRA).

#### Introduction

This section describes how user accounts in Novell eDirectory can be managed with User Management Resource Administrator (UMRA). To manage user accounts in Novell eDirectory, the LDAP protocol is used.

The main functions of UMRA to manage Novell eDirectory user accounts are:

- Create user account
- Set user account password
- Manage user account attributes
- Delete user accounts
- Setup user account group memberships

All of these functions are described in this document. Sample projects that implement these functions are available from the Tools4ever web-site at <http://www.tools4ever.com> <http://www.tools4ever.com>.

The sample projects are implemented and tested in the following environment:

Novell Netware 5.60

Novell eDirectory 8.6.0

The LDAP protocol supports secure (SSL) and not secure sessions. UMRA supports both mechanisms. Because of confidentiality requirements, in most environments, the secure SSL implementations will be used. This chapter includes a section how to setup a secure LDAP environment between the UMRA software and the Novell eDirectory environment.

#### Secure LDAP eDirectory environment

To setup a secure LDAP environment, certificates must be configured on both the computer that runs the UMRA software and the Novell eDirectory server. The certificates must be signed by a Certification Authority that is trusted by both sides.

### SSL Certification Authority

By default, Novell eDirectory installs and configures an SSL Certification Authority and the eDirectory LDAP Server can be configured to use a certificate from the Certification Authority. A simple method to setup the certificates for both sides is to export the root certificate from the Novell eDirectory Certification Authority and import that certificate on the computer that runs the UMRA software. This procedure is described step by step in the following section:

### Enabling SSL on Novell eDirectory LDAP Server

Start the Novell management application ConsoleOne and locate the LDAP Server item in eDirectory.



Figure 6: eDirectory shown in Novell ConsoleOne with the LDAP Server on server SRVNW6.

Access the properties of the LDAP Server. Several attributes deal with the configuration of the SSL support of the LDAP Server.

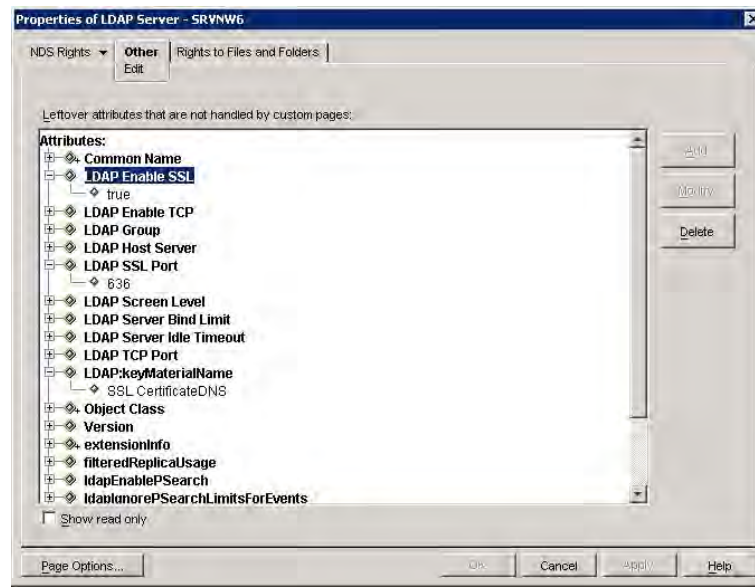


Figure 7: Novell ConsoleOne: LDAP Server SSL attributes.

1. **LDAP Enable SSL:** Set to true to enable SSL support for the Novell eDirectory server.
2. **LDAP SSL Port:** The TCP port used to access the LDAP Server. Default LDAP SSL port: 636.
3. **LDAP:keyMaterialName:** The name of the SSL certificate used by the LDAP Server. By default, a certificate is specified that is issued by the Certification Authority of the Novell eDirectory server.

By default, the SSL support is enabled on port 636 and a certificate is configured. If you want to use a different port or certificate, you need to update the attributes.

### Exporting the Novell eDirectory root certificate

To export the root certificate, select the LDAP Server certificate from eDirectory.



Figure 8: Novell ConsoleOne: LDAP Server SSL attributes.

Access the properties of the certificate and navigate to the **Trusted root certificate** of the certificate. Click **Export**.

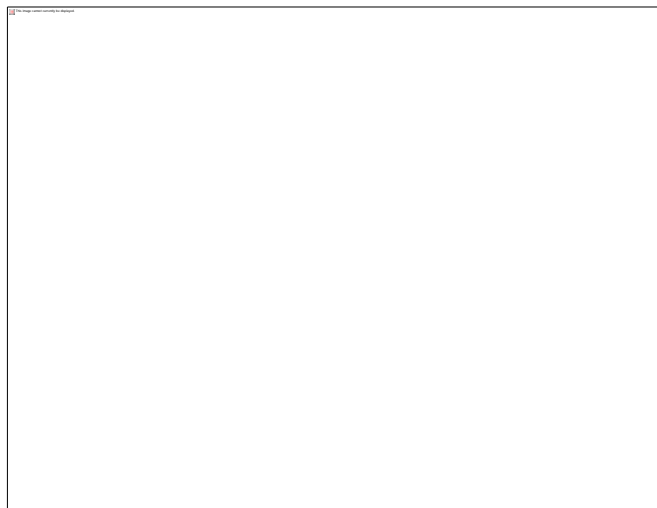


Figure 9: Novell ConsoleOne: Properties of trusted root certificate.

Follow the instruction to export the root certificate to a file in binary DER format. Do not include the private key of the certificate.

### Importing the certificate on the UMRA computer

To complete, you need to import the exported certificate on the computer that runs the LDAP Client, e.g. the UMRA software. The UMRA software that communicates with the LDAP Server is either the **UMRA Console** application or the **UMRA Service**. For each software module, the procedure to import the certificate is different. For the UMRA Console application, the certificate is imported for the logged on user account. For the UMRA Service, the certificate must be imported for the computer that runs the service.

### Importing the certificate for the UMRA Console

In Microsoft Internet Explorer 6, select **Tools, Internet options...**, **Content, Certificates**. Press **Import...** and follow the instructions of the wizard. When asked, select the option *Automatically select the certificate store based on the type of certificate*. When completed, you can check the list with **Trusted Root Certification Authorities**. The list must contain the new entry.

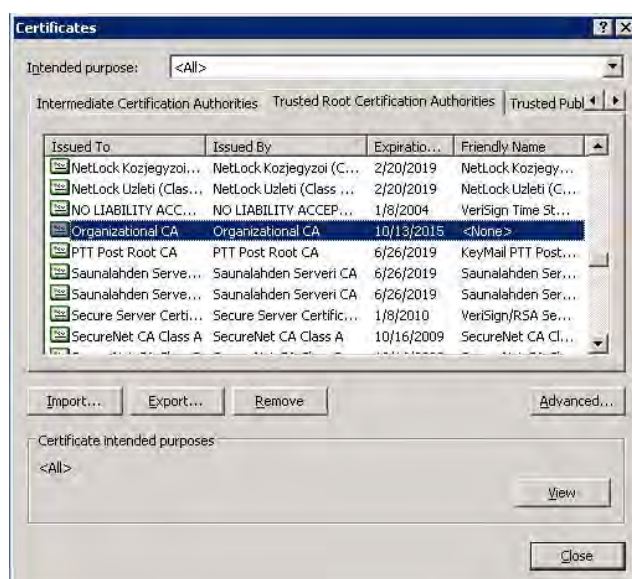


Figure 10: List with Trusted Root Certification Authorities showing the imported LDAP Server certificate

### Importing the certificate for the UMRA Service

On the computer that runs the **UMRA Service**, start the **Microsoft Management Console** by selecting menu option **Start, Run**. Enter **MMC** and press Enter. Add the management snap-in to manage certificates with menu option **File, Add/Remove snap-in**. Press **Add** and select snap-in **Certificates**.

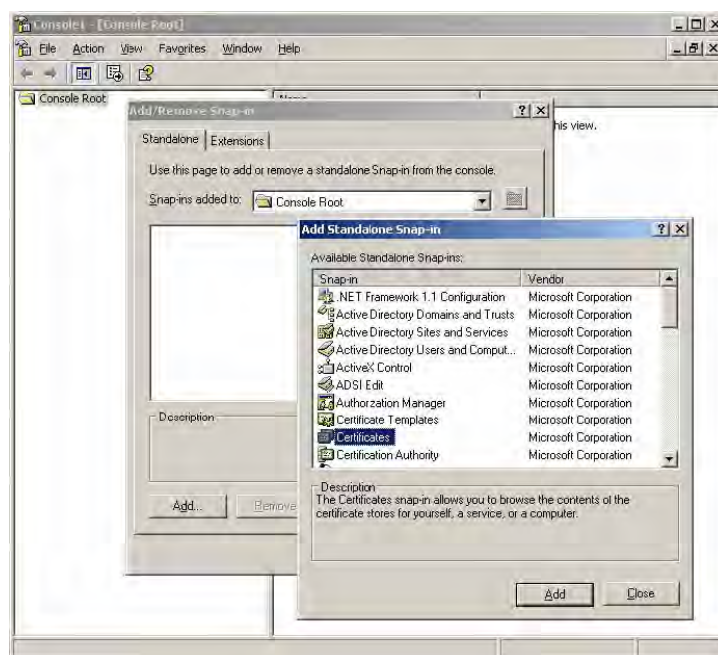


Figure 11: Add Certificates snap-in to Microsoft Management Console in order to import the certificate for UMRA Service.

Click **Add** and select the option to manage certificates for the **Computer account**. Next select the **Local computer** and exit the configuration dialogs. With the MMC you can now manage the certificates of the local computer.

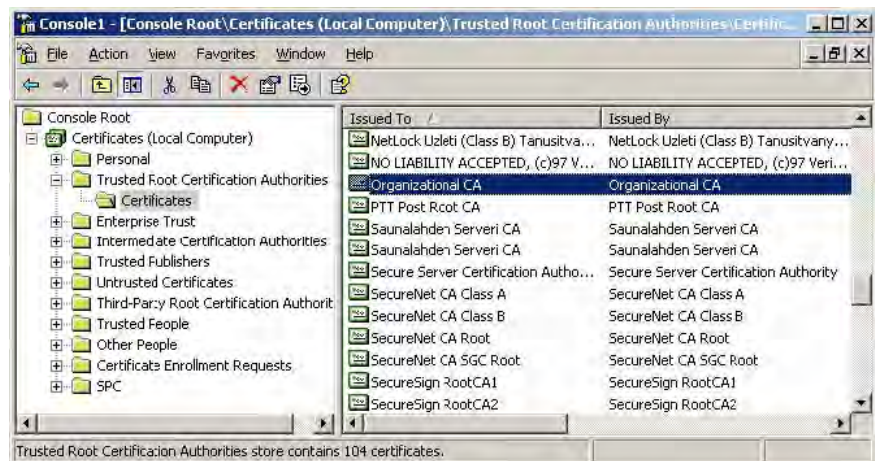


Figure 12: The MMC configured to manage the certificates of the local computer as used by the UMRA Service.

To add the certificate, browse to the item **Certificates** of the **Trusted Root Certification Authorities** and select menu option **All tasks, Import...** and follow the instructions of the wizard. When asked, select the option *Automatically select the certificate store based on the type of certificate*. When completed, you can check the list with **Trusted Root Certification Authorities**. The list must contain the new entry.



### Testing the certificate configuration

You can test the SSL configuration with the tool LDP.EXE, part of the **Windows Support Tools** from Microsoft Windows Server 2003. (Note: the LDP.EXE tool part of the Windows Support Tools from Microsoft Windows 2000 does not support SSL). When the Windows Support Tools for Microsoft Windows Server 2003 are installed, start the tool by entering LDP.EXE on the command prompt. Select menu option **Connection, Connect....** Specify the connection settings and enable SSL.

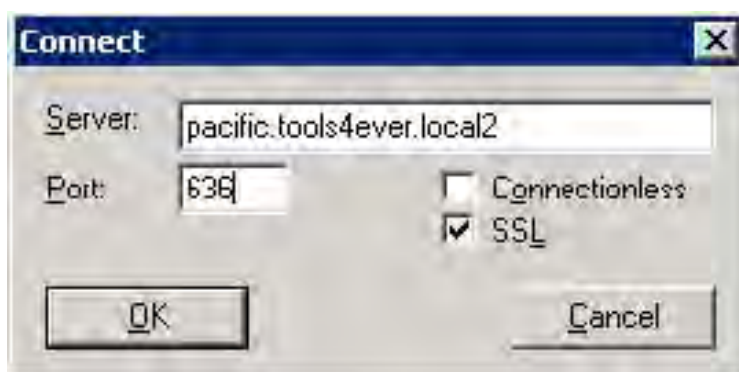


Figure 13: LDAP.EXE connection settings

When the SSL certificates are not installed successfully, the connection cannot be established.

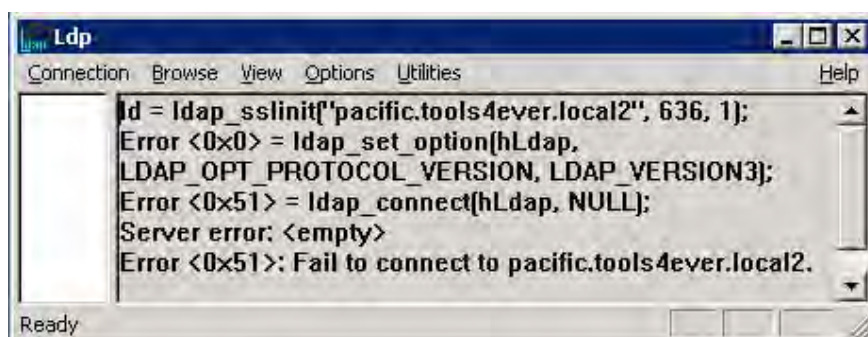


Figure 14: LDP.EXE failure when SSL certificates are not or incorrectly configured.



When the SSL are correctly installed, the connection is established with the LDAP Server.

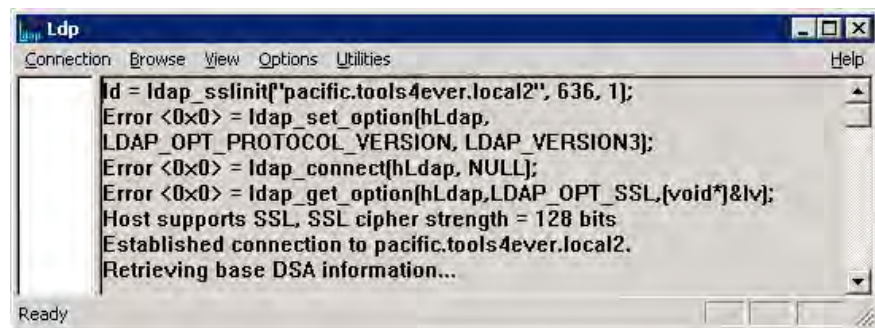


Figure 15: LDP.EXE successful connection setup using SSL.

When successfully configured, the UMRA software can communicate with the LDAP Server using SSL.

### Creating user accounts in Novell eDirectory

This example describes a mass project that is used to import a number of user accounts from a csv-file into Novell eDirectory. The script of the project is deliberately limited to the essential actions that deal with user account creation in Novell eDirectory with UMRA using LDAP. A similar script can be used with UMRA Form and Automation projects.

### Example project

The example project can be found at the following location relative to the UMRA Console program directory:

.\Example

Projects\LDAP\Novell\AddUserMass\Novell\_eDir\_CreateUserAccountM  
ass.upj

The example project contains embedded input data representing user accounts. For each line of the input data, the script does the following:

1. Setup a secure LDAP session with the LDAP Server;
2. Setup the LDAP modification data to add the user account;
3. Add the account.

The following section describes the project in detail.

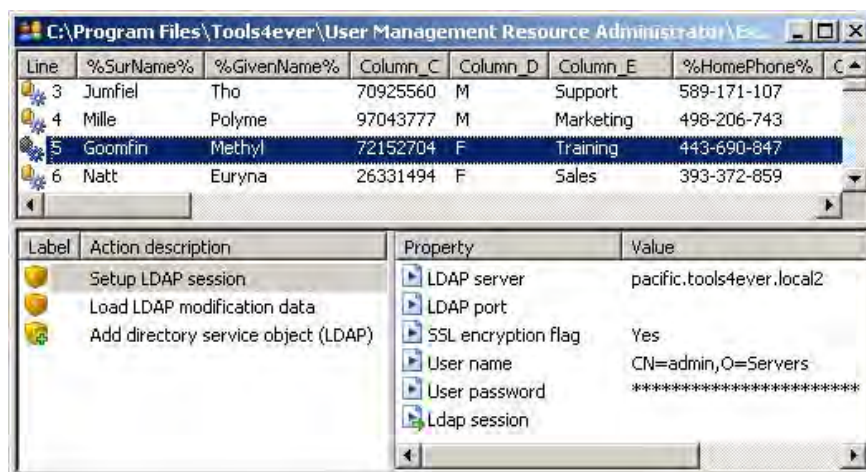


Figure 16: Example project to import bulk user accounts in Novell eDirectory using UMRA and secure LDAP (SSL).

#### Setting up an LDAP session

The LDAP session is setup with the LDAP Server, in this case the computer that runs Novell eDirectory: *pacific.tools4ever.local2*.



Figure 17: Setup LDAP session script action

The **LDAP server** is specified using a DNS name or TCP/IP address. The **LDAP port** only needs to be specified when it does not equal the default port (LDAP, no SSL: 389, LDAP with SSL: 636). The **SSL encryption flag** is set to enable secure communication. When SSL is used, certificates need

to be installed on both the LDAP Server and Client side. The **User name** depends on the directory service implementation. In this case, an organization **O=Servers** contains the administrator account **admin** that is used to access the data. The password is not actually shown.

When the action is successfully executed, the session is initialized. The session object is stored in a variable with default name: **%LdapSession%**. This session variable is used in subsequent actions of the script.

Note: When the action is executed, the password specified is send over the line. When SSL is enabled, the password is automatically encrypted since all communication with the LDAP Server is encrypted. When SSL is not used, the password is send as clear text.

#### *Loading LDAP modification data*

With the next action, the data structure used to add the user account is prepared. This data structure contains a number of attributes, each with one or more values. The exact attributes used to add a user account vary for each directory service implementation that supports LDAP.

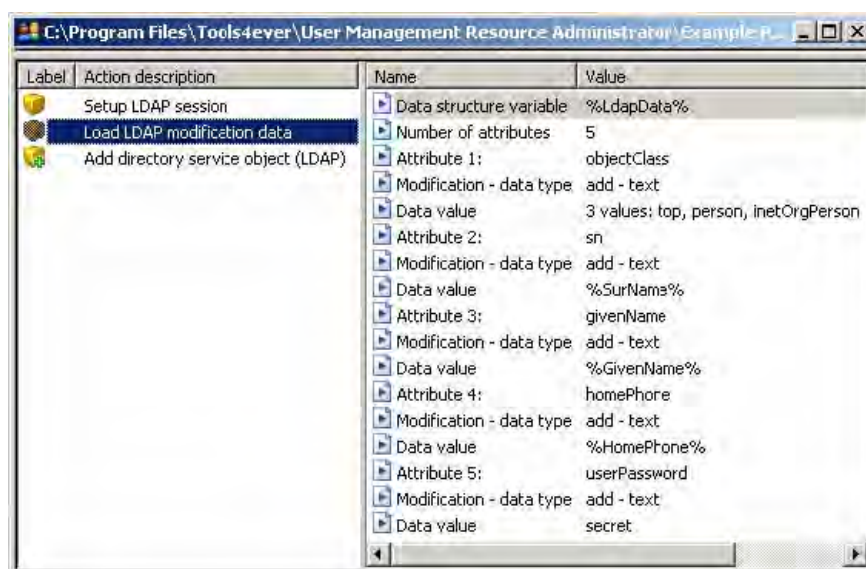


Figure 18: Load LDAP modification data script action.

The resulting data structure is stored in a variable. In this example, the default variable name **%LdapData%** is used to store the structure. The variable is used in subsequent script actions.

According to the Novell eDirectory schema documentation, a user account must have the following attributes defined:

1. **objectClass**: This attribute must get 3 values, **top**, **person** and **inetOrgPerson** to make the new object a user account.
2. **cn** (Common Name): The common name is the unique identifier of the object in the directory service.
3. **sn** (Surname): The surname attribute must have a text value, representing the last name of the user account.

The common name is defined in the next action (Add directory service object (LDAP)). The **objectClass** and **sn** attribute are initialized in the modification data structure. Besides these attribute, also the **givenName**, **homePhone** and **userPassword** attributes are setup.

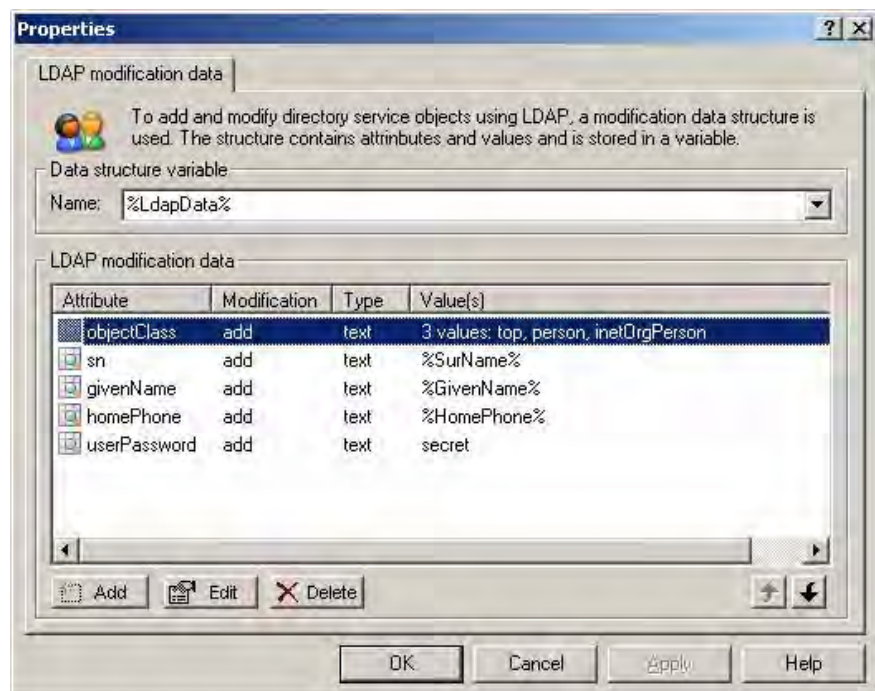


Figure 19: Properties of action to setup LDAP modification data.

Depending on the schema definition, an attributes can have a single or multiple values. Values can be specified using variables. In the example shown, the attributes **sn**, **givenName** and **homePhone** are specified using variables **%SurName%**, **%GivenName%** and **%HomePhone%** and linked to the input csv file. The **objectClass** attribute gets the same 3 values for each user account: **top**, **person** and **inetOrgPerson** as defined by the eDirectory schema.

To setup an individual attribute, double click one of the attributes.



*Figure 20: Dialog used to specify the modification type and values of a single attribute.*

### *Add directory service object*

Finally the user account is created with script action **Add directory service object (LDAP)**. During this action, the UMRA software actually communicates with the LDAP Server.

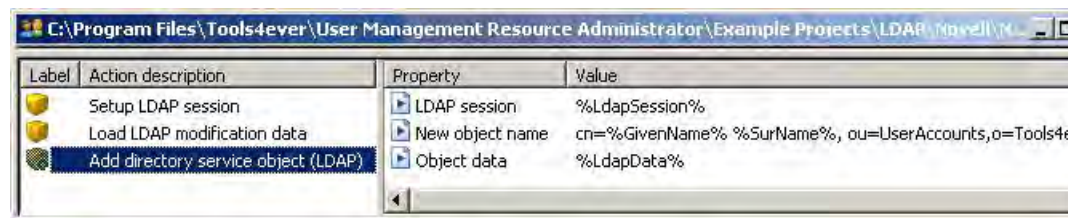


Figure 21: Add directory service object (LDAP) script action.

The action has 3 attributes:

1. **LDAP session:** The session is specified by a variable and must have been initialized with a previously executed action **Setup LDAP session**.
2. **New object name:** The common name of the new LDAP object. The name must be specified as a distinguished name and depends on the directory service structure. In this example, the account is created in organizational unit **UserAccounts**, part of the organization **Tools4ever** of the eDirectory tree. The first part of the new name is composed from the 2 variables **%GivenName%** and **%SurName%**. The total name must be unique in the directory service.
3. **Object data:** The modification data structure that is the result of script action **Load LDAP modification data**. The data is referenced using a variable, **%LdapData%** in this example.

### **Log information**

When executed successfully, the user account is created. The log file of a successfully executed user account creation script is shown below:

Starting User Management Resource Administrator session, build 1207 at  
11:58:45 11/18/2005

11:58:45 11/18/2005 \*\*\*\*\* Processing entry 241...

11:58:45 11/18/2005 Variable 1: %GivenName%=Circe

11:58:45 11/18/2005 Variable 2: %SurName%=Eris

11:58:45 11/18/2005 Variable 3: %HomePhone%=460-608-205

11:58:45 11/18/2005 Variable 4: %NowDay%=18

11:58:45 11/18/2005 Variable 5: %NowMonth%=11

11:58:45 11/18/2005 Variable 6: %NowYear%=2005

11:58:45 11/18/2005 Variable 7: %NowHour%=11

11:58:45 11/18/2005 Variable 8: %NowMinute%=58

11:58:45 11/18/2005 Variable 9: %NowSecond%=45

11:58:45 11/18/2005 Setting up LDAP sessions with host  
'pacific.tools4ever.local2'. Using SSL encryption: 'Yes'.

11:58:45 11/18/2005 User name: 'CN=admin,O=Servers'.

11:58:45 11/18/2005 Secure LDAP session established with host  
'pacific.tools4ever.local2' (Protocol: 'SSL 3.0 client-side', encryption: 'RC4  
stream', cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA',  
2048 bits).

11:58:45 11/18/2005 Authenticating user 'CN=admin,O=Servers'...

11:58:45 11/18/2005 User 'CN=admin,O=Servers' successfully authenticated on  
LDAP server host 'pacific.tools4ever.local2'.

11:58:45 11/18/2005 LDAP session information stored in variable  
'%LdapSession%'.

11:58:45 11/18/2005 Storing LDAP modification data in variable '%LdapData%'.

11:58:45 11/18/2005 LDAP modification data:

11:58:45 11/18/2005 \*\*\*\*\* Modification data element: 0  
\*\*\*\*\*

11:58:45 11/18/2005 Operation: 'add', type of data: 'text'

11:58:45 11/18/2005 Attribute: 'objectClass'

11:58:45 11/18/2005 Value 0: 'top'

11:58:45 11/18/2005 Value 1: 'person'

11:58:45 11/18/2005 Value 2: 'inetOrgPerson'

11:58:45 11/18/2005 \*\*\*\*\* Modification data element: 1  
\*\*\*\*\*

11:58:45 11/18/2005 Operation: 'add', type of data: 'text'

11:58:45 11/18/2005 Attribute: 'sn'

11:58:45 11/18/2005 Value 0: 'Eris'

11:58:45 11/18/2005 \*\*\*\*\* Modification data element: 2  
\*\*\*\*\*

11:58:45 11/18/2005 Operation: 'add', type of data: 'text'

11:58:45 11/18/2005 Attribute: 'givenName'

11:58:45 11/18/2005 Value 0: 'Circe'

11:58:45 11/18/2005 \*\*\*\*\* Modification data element: 3  
\*\*\*\*\*

11:58:45 11/18/2005 Operation: 'add', type of data: 'text'

11:58:45 11/18/2005 Attribute: 'homePhone'

11:58:45 11/18/2005 Value 0: '460-608-205'

11:58:45 11/18/2005 \*\*\*\*\* Modification data element: 4  
\*\*\*\*\*

11:58:45 11/18/2005 Operation: 'add', type of data: 'text'

11:58:45 11/18/2005 Attribute: 'userPassword'

11:58:45 11/18/2005 Value 0: 'secret'

11:58:45 11/18/2005 Adding LDAP directory service object 'cn=Circe Eris,  
ou=UserAccounts,o=Tools4ever' with LDAP modification data obtained from  
variable '%LdapData%'.

11:58:45 11/18/2005 \*\*\*\*\* Ready processing entry 241...

11:58:45 11/18/2005 Total number of script action execution errors: 0.

End of session



The log file shows the following information:

1. The LDAP session is setup using SSL with Novell eDirectory host **pacific.tools4ever.local2**.
2. The account used to authenticate and access the LDAP Server is **cn=admin, o=Servers**. The password specified is sent encrypted over the line since SSL is enabled.
3. Once the connection is successfully initialized, the SSL encryption parameters are shown. The user is authenticated and the session structure is stored in variable %LdapSession%.
4. The LDAP modification data structure is setup and stored.
5. The item is added to the directory service, e.g. the user account is created.

#### **Setting a user account password on Novell eDirectory**

This example describes a form project to reset the password of a selected user account. With respect to the LDAP call, the project scripts include the following features:

- Replace the attribute value of an existing attribute (password)
- Search user accounts in LDAP

The project is intentionally implemented as simple as possible to show the usage of LDAP script actions in the best possible way.

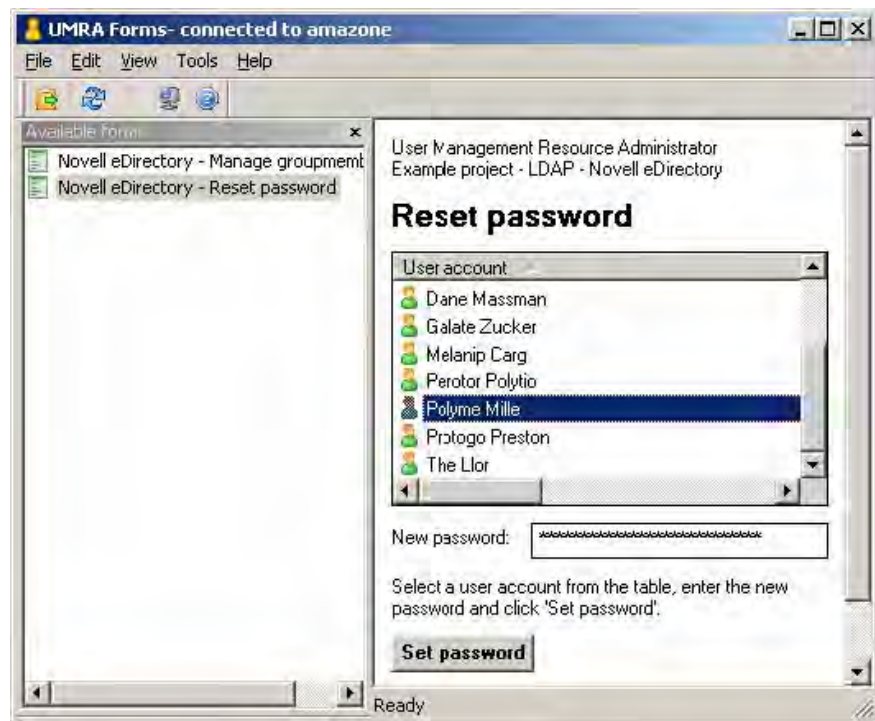


Figure 22: UMRA Forms application to reset passwords of Novell eDirectory user accounts.

The result form shows a list with user accounts. The end-user can select an account from the list and specify a new password. Next, the password is set when the end-user clicks the **Set password** button.

#### Example project

The UMRA application consists of the following projects, described in detail in the next sections. The example project can be found at the following location relative to the UMRA Console program directory:

.\\Example Projects\\LDAP\\Novell\\ResetPassword

The project directory contains all three projects of the UMRA application.

Project	Description
ResetPassword - InitializeVars	Simple project with a script only to initialize LDAP session and environment variables that are used by the other projects.
ResetPassword - GetUsers	Project with a script only to collect the user accounts with an LDAP search and store the results in a table.
Novell eDirectory - Reset password	The main project that contains the form of the UMRA application and the script to reset the password of the selected user account.

*Table 6: Projects of UMRA application to reset the password of Novell eDirectory user accounts.*

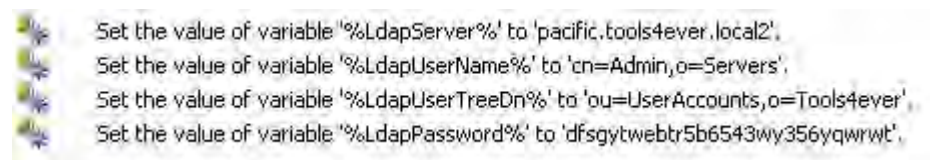
#### UMRA application flow

This section briefly describes the most important features executed when the UMRA application runs:

1. In the UMRA Forms application, the end-user selects the form project - **Novell eDirectory - Reset password**. A request is sent to the UMRA Service to create the form and contents of the form and return it to the UMRA Forms client.
2. As a response, the UMRA Service checks the access rights and loads the form of the **Novell eDirectory - Reset password** project. As the initial project, the **ResetPassword - GetUsers** is configured. Before the form is created, the script of this project is executed.
3. The **ResetPassword - GetUsers** project executes the variable initialization project **ResetPassword - InitializeVars**. Next, the project connects to the LDAP server to setup a session, and queries for the user accounts. The results are stored in a table and shown in the form.
4. The end-user selects a user account, enters a new password and submits the form.
5. The script of the main project, **Novell eDirectory - Reset password** is now executed. First, the variables are initialized (project: **ResetPassword - InitializeVars**). An LDAP modification data structure that contains the new password is initialized. The password is updated. The procedure contains with step 2.

### Project: ResetPassword - InitializeVars

The project only contains a script, not a form. The script initializes the variables that are specific to your environment. The other projects used in the UMRA application do not contain any other environment dependant variable.



*Figure 23: Script actions of initialization project **ResetPassword - InitializeVars**.*

The variables initialized are used to setup the LDAP session (%LdapServer%, %LdapUserName and %LdapPassword%) and to search for user accounts in the eDirectory subtree (%LdapUserTreeDn%).

The project is used in the UMRA application at two places:

1. As the initial project of project **ResetPassword - GetUsers**;
2. The project is also executed as the first line of the script of the main project **Novell eDirectory - Reset password** to initialize the variables.

Note that the password specified is encrypted. The actual value is obtained by decryption just before the actual session is setup.

### Project: ResetPassword - GetUsers

The project contains a script only, not a form. **The project is configured as the initial project of the main project.** The script is therefore executed by the UMRA Service, just before the form is created. Goal of the project is to return a variable (%LdapUsers%) that holds a table with all user accounts.

The script of the project establishes a connection with the LDAP Server to perform a query to find user accounts. The accounts are stored in a table variable.



Figure 24: Script action to execute script of project ResetPassword - InitializeVars.

Before the LDAP session is setup, the variables used are initialized. Next, the LDAP session is established.

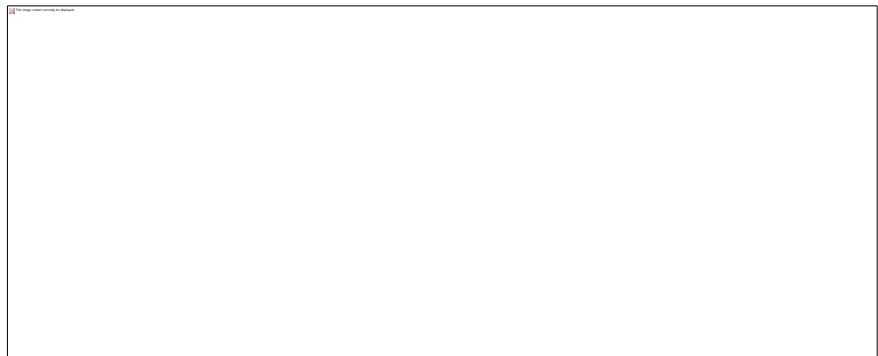
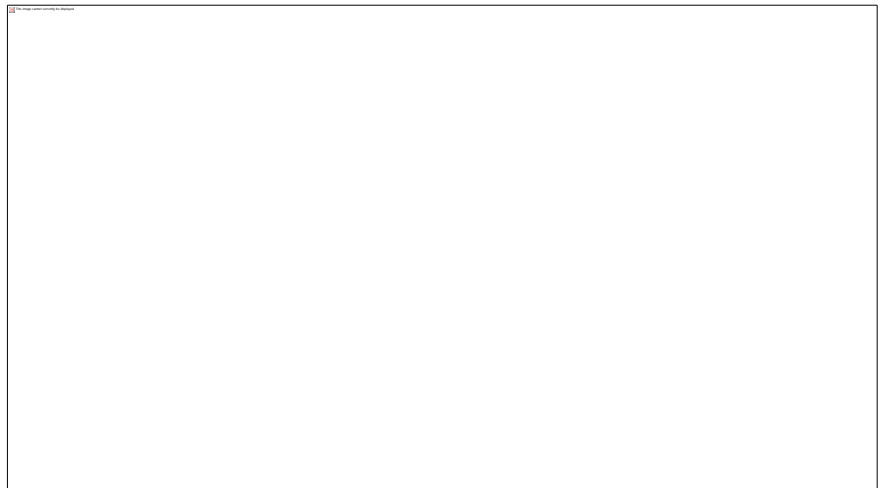


Figure 25: Script action to setup LDAP session.

In this case, the session is encrypted so all communication between the UMRA software and the LDAP Server is secure. The resulting LDAP session is stored in the **LDAP session** property output variable **%LdapSession%**.



*Figure 26: Script action to search in Novell eDirectory.*

Finally, the actual search is executed. The search uses the established session (variable: %LdapSession%) to communicate with the LDAP Server. The results of the search are stored in table variable **%LdapUsers%**. Note that the variable contains all of the table data.

In the subtree, specified by %LdapUserTreeDn% (example: ou=UserAccounts,o=Tools4ever), the search operation collects all directory service items of type **User** (Filter: objectClass=User). For each matching item, e.g. all user accounts, the common name (**cn**) and the distinguished name is returned. So the final table contains 2 columns. The common name is used to present the end-user a user-friendly name while the distinguished name is used to refer to the actual account.

**Project: Novell eDirectory - Reset password**

The main project of the UMRA application contains both a form and a script. The form shows a list with user accounts and a text input box to specify the new password for the selected user account.



The image shows a 'Form preview' window titled 'Form preview' with a standard Windows-style title bar. Inside the window, the text 'User Management Resource Administrator' and 'Example project - LDAP - Novell eDirectory' is displayed at the top. Below this is the main title 'Reset password' in a large, bold font. A list box labeled 'User account' contains several entries, each preceded by a small green icon: 'Aglates Sisold', 'Circe Eris', 'Dane Massman', 'Galate Zucker', 'Melanip Carg' (which is highlighted with a blue background), 'Perotor Polytio', and 'Polyme Mille'. Below the list box is a text input field labeled 'New password:' containing the text 'skoksk'. At the bottom of the form, there is a button labeled 'Set password'. Below the button, a small instruction reads: 'Select a user account from the table, enter the new password and click 'Set password'.'

*Figure 27: Preview in UMRA Console of project Novell eDirectory - Reset password*

When the **Set password** button is pressed, the form is submitted to the UMRA Service and the form project is executed.

### Table with user accounts

Besides some text elements, the project form contains a table, password input field and a submit button.

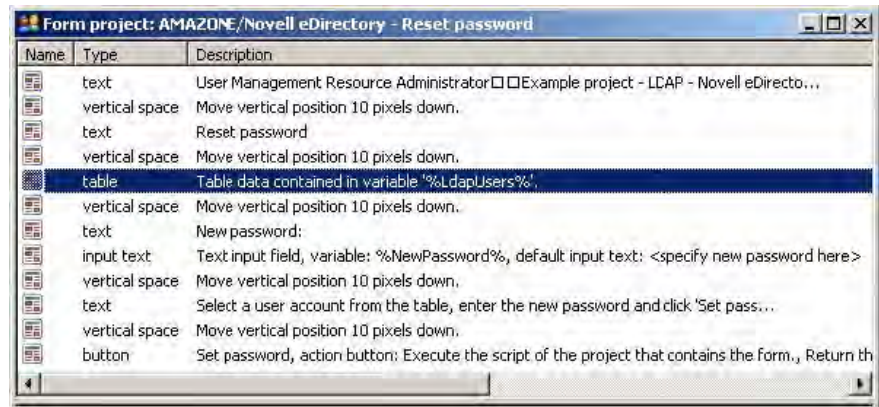


Figure 28: Definition of the form of project Novell eDirectory - Reset password



The table shows the user accounts, collected with project **ResetPassword - GetUsers**. The table data is stored in variable **%LdapUsers%**. When setting up the form generic table, note that the columns need to be defined for the table. The variable **%LdapUsers%** stores the table data, but not the name of the table columns.

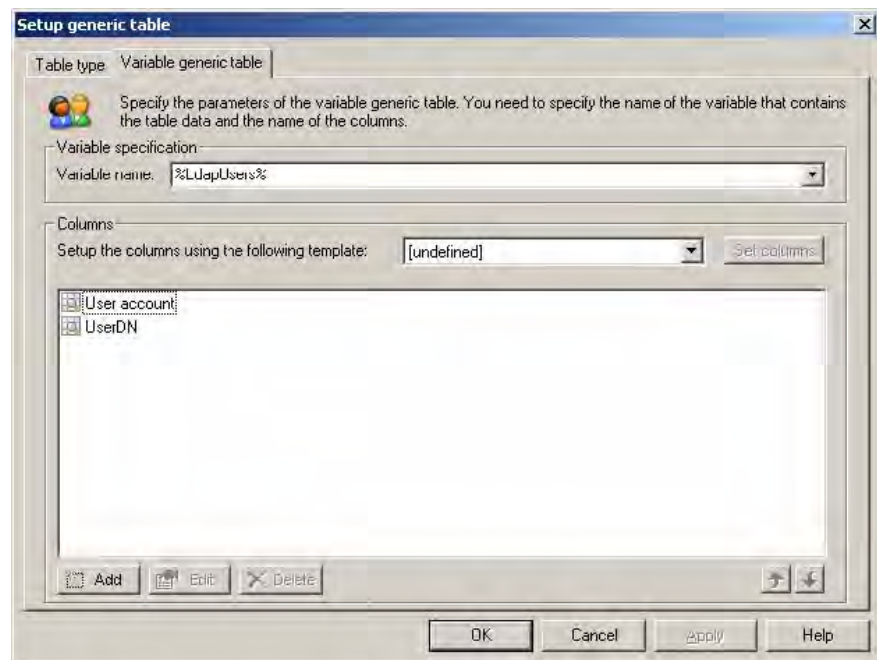


Figure 29: Specification of columns of generic table derived from variable **%LdapUsers%**.

In order to use the table data successfully, you need to know the number of columns and the name and type of data for each column. You must enter the name of the columns manually. In the example shown, the columns are called **User account** and **UserDN**, referencing the name of the user account and the distinguished name. When configuring the columns for the form, the width of the column with the distinguished name is set to 0.

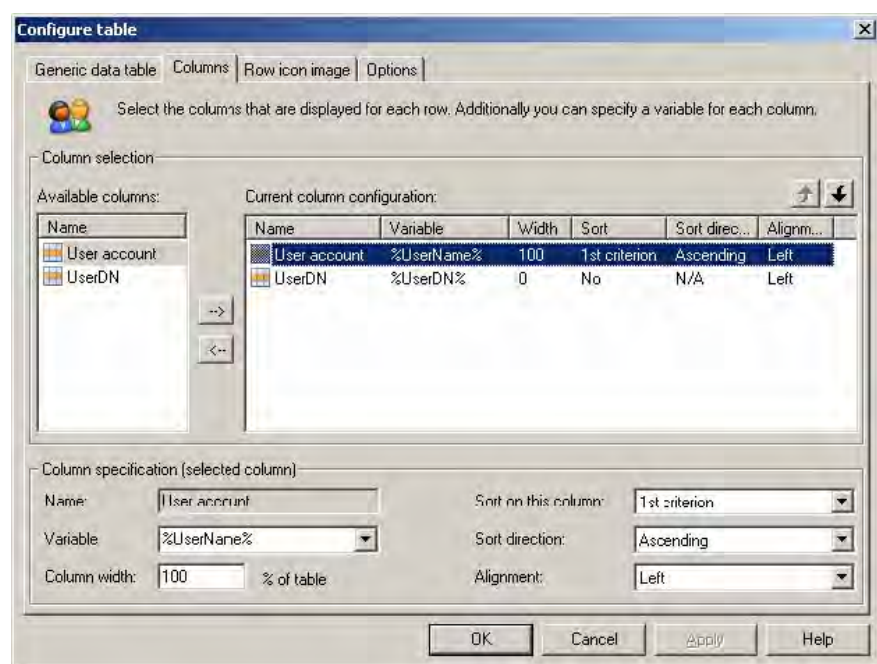


Figure 30: Specification of the table columns shown in the form.

With this mechanism, you can return the name of the selected user account in a variable (%UserDN%) but not show the (ugly format) name to the end-user. Instead, the common name is shown.

### Setting a password submit button

When the **Set password** button is pressed, the form output variables are collected and send to the UMRA Service. The form output variables are:

1. **%UserName%**: The user-friendly name of the selected user account. This variable is returned only for logging purposes;

2. **%UserDN%**: The name that uniquely identifies the selected user account. The variable is used to identify the directory service user account of which the password is reset.
3. **%NewPassword%**: The new password specified for the selected user account.

The variables form the input of the project's script.

### Script to reset a password

The script to reset the password of the selected user account connects to the LDAP Service, initializes the attributes modification data and updates the directory service user account.



Figure 31: Script action to execute the script of project ResetPassword - InitializeVars.

With the **Execute script** action, the variables used to setup the LDAP session are initialized. The LDAP session is initialized using SSL.



Figure 32: Script action to setup the LDAP session.

The LDAP modification data is setup with a single attribute that is used for the user account password. The name of the attribute, as defined by

the eDirectory schema is: **userPassword**. The type of modification is: **replace** since the current attribute value must be replaced by the new one.

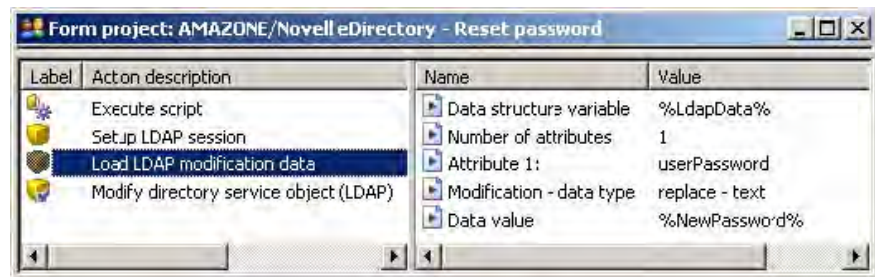


Figure 33: Script action to initialize the LDAP modification data.

The value of the attribute is set equal to the specified variable **%NewPassword%**.

When the modification data is setup, the update can be made.



Figure 34: Script action to modify the directory service item.

The action is completely specified by the following variables:

1. **%LdapSession%**: The data structure representing the LDAP session with the LDAP Server.
2. **%UserDN%**: The distinguished name of the of the selected user account. The name identifies the directory service item, e.g. the user account.
3. **%LdapData%**: The modification data, holding the new replacement value of the password attribute of the user account.

When the script is executed, the form is returned to allow the end-user to reset the password of the next user account.

### Deleting user accounts in Novell eDirectory

To delete user accounts or other directory service items, a very simple UMRA script is required. The script contains two action only:

1. **Setup LDAP session** on page 5: Connect to the LDAP Server and authenticate the connecting eDirectory user account;
2. **Delete directory service object (LDAP)** on page 11: Delete the directory service object or item. The object to delete must be specified by its distinguished name. Example: cn=John Smith, ou=SomeOU, o=Tools4ever.

To delete an item from the directory service, no attribute modification structure is required. When the directory service item is deleted, all attributes of the item are deleted automatically.

### Setting up user account group memberships on Novell eDirectory

To setup user account group memberships on Novell eDirectory, you need to update the attributes of two directory service items: the user account and the group. This is specified by the eDirectory service schema.

The following table shows the attribute updates in order to add a user account to a group:

Item	Attribute	Value update action
User account	groupMembership	Add distinguished name of group
User account	securityEquals	Add distinguished name of group
Group	uniqueMember	Add distinguished name of user account
Group	equivalentToMe	Add distinguished name of user account.

*Table 7: Required attribute value changes to update group membership in Novell eDirectory.*

The example project is not very user-friendly but shows exactly how to use the LDAP script actions. The example project can be found at the following location, relative to the UMRA Console program directory:

.\\Example Projects\\LDAP\\Novell\\AddToGroup\\Novell eDirectory - Add User To Group.ufp

The UMRA application consists of a single project with a form and script. The form show some text fields and two input fields for the distinguished names of the user account and the group.



Figure 35: Form to enter the distinguished names of the user accounts and group.

A more user-friendly form is available from the example project, described in the next topic.

The values entered in the form input fields are stored in the variables **%UserDN%** and **%GroupDN%**.

When the end-user clicks the **Add** button, the script of the project is executed. The script first initializes the session with the LDAP Server. Next, the modification data to update the user account attributes are initialized.

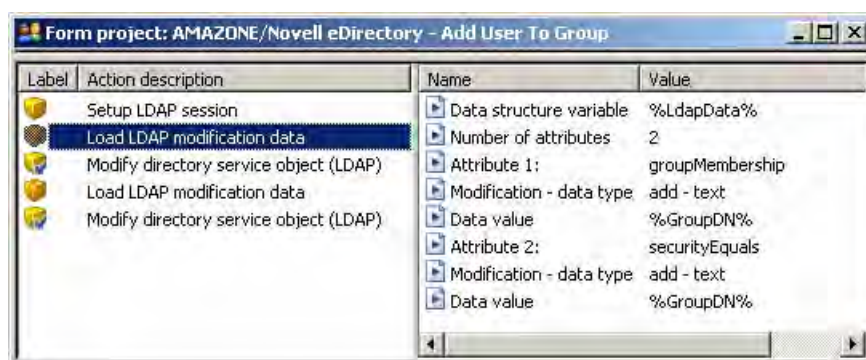


Figure 36 Script action to initialize the LDAP modification data to update user account attributes groupMembership and securityEquals.

Two attributes of the user account, groupMembership and securityEquals are updated by adding the value of the distinguished name of the group (%GroupDN%).

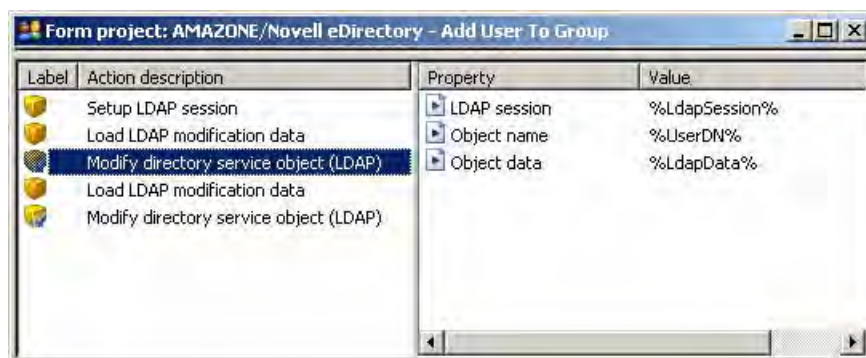


Figure 37: Script action to update the attributes of the user account.

Next, the attributes of the group are updated.

This time, two attributes of the group, uniqueMember and equivalentToMe are updated by adding the value of the distinguished



name of the user account (%UserDN%).

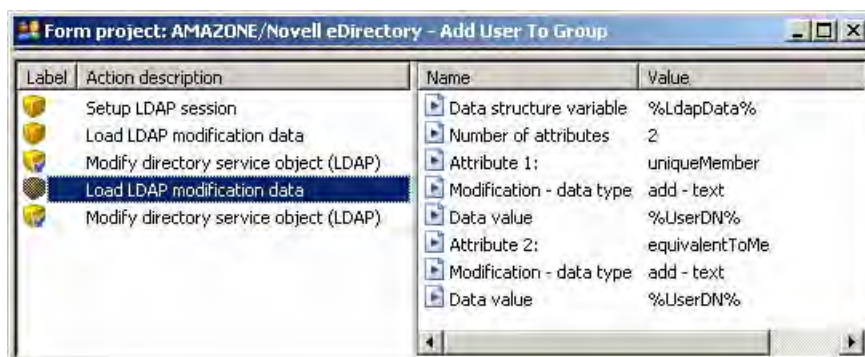


Figure 38 Script action to initialize the LDAP modification data to update group attributes uniqueMember and equivalentToMe.

When the last action is executed successfully, the user account has become a member of the group.

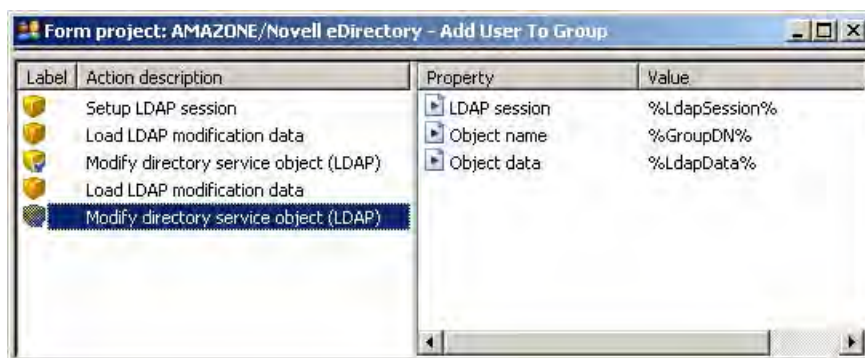


Figure 39: Script action to update the attributes of the group.

If the last action fails, it is advised to remove the values from the user account attributes that were added in the previous modification action. To keep the script clean and clear, this action is not part of the example script.

The UMRA Service log file shows all of the action executed.

#### UMRA Service log

```
17:40:23 11/24/2005 Variable 1: %UserDN%=cn=Melanip Carg,
ou=UserAccounts,o=Tools4ever
```



17:40:23 11/24/2005 Variable 2: %GroupDN%=cn=SupportGroup,  
o=Tools4ever

17:40:23 11/24/2005 Variable 3:  
%UmraFormSubmitAccount%=T4ELOC2\Administrator

17:40:23 11/24/2005 Variable 4: %NowDay%=24

17:40:23 11/24/2005 Variable 5: %NowMonth%=11

17:40:23 11/24/2005 Variable 6: %NowYear%=2005

17:40:23 11/24/2005 Variable 7: %NowHour%=17

17:40:23 11/24/2005 Variable 8: %NowMinute%=40

17:40:23 11/24/2005 Variable 9: %NowSecond%=23

17:40:23 11/24/2005 Variable 10: %LdapSession%=(0,0X0)

17:40:23 11/24/2005 Variable 11: %LdapData%=(0,0X0)

17:40:23 11/24/2005 Setting up LDAP sessions with host  
'pacific.tools4ever.local2'. Using SSL encryption: 'Yes'.

17:40:23 11/24/2005 User name: 'cn=Admin,O=Servers'.

17:40:23 11/24/2005 Secure LDAP session established with host  
'pacific.tools4ever.local2' (Protocol: 'SSL 3.0 client-side', encryption: 'RC4  
stream', cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA',  
2048 bits).

17:40:23 11/24/2005 Authenticating user 'cn=Admin,O=Servers'...

17:40:23 11/24/2005 User 'cn=Admin,O=Servers' successfully authenticated on  
LDAP server host 'pacific.tools4ever.local2'.

17:40:23 11/24/2005 LDAP session information stored in variable  
'%LdapSession%'.

17:40:23 11/24/2005 Storing LDAP modification data in variable '%LdapData%'.

17:40:23 11/24/2005 LDAP modification data:

17:40:23 11/24/2005 \*\*\*\*\* Modification data element: 0  
\*\*\*\*\*

17:40:23 11/24/2005 Operation: 'add', type of data: 'text'

17:40:23 11/24/2005 Attribute: 'groupMembership'

17:40:23 11/24/2005 Value 0: 'cn=SupportGroup, o=Tools4ever'

17:40:23 11/24/2005 \*\*\*\*\* Modification data element: 1  
\*\*\*\*\*

17:40:23 11/24/2005 Operation: 'add', type of data: 'text'

17:40:23 11/24/2005 Attribute: 'securityEquals'

17:40:23 11/24/2005 Value 0: 'cn=SupportGroup, o=Tools4ever'

17:40:23 11/24/2005 Modifying LDAP directory service object 'cn=Melanip Carg, ou=UserAccounts,o=Tools4ever' with LDAP modification data obtained from variable '%LdapData%'.

17:40:23 11/24/2005 Storing LDAP modification data in variable '%LdapData%'.

17:40:23 11/24/2005 LDAP modification data:

17:40:23 11/24/2005 \*\*\*\*\* Modification data element: 0  
\*\*\*\*\*

17:40:23 11/24/2005 Operation: 'add', type of data: 'text'

17:40:23 11/24/2005 Attribute: 'uniqueMember'

17:40:23 11/24/2005 Value 0: 'cn=Melanip Carg,  
ou=UserAccounts,o=Tools4ever'

17:40:23 11/24/2005 \*\*\*\*\* Modification data element: 1  
\*\*\*\*\*

17:40:23 11/24/2005 Operation: 'add', type of data: 'text'

17:40:23 11/24/2005 Attribute: 'equivalentToMe'

17:40:23 11/24/2005 Value 0: 'cn=Melanip Carg,  
ou=UserAccounts,o=Tools4ever'

17:40:23 11/24/2005 Modifying LDAP directory service object 'cn=SupportGroup, o=Tools4ever' with LDAP modification data obtained from variable '%LdapData%'.

17:40:24 11/24/2005 Form message:

'11/24/2005,17:40:23,T4ELOC2\Administrator,"Form submit",OK,"Novell eDirectory - Add User To Group"'

### Managing user account group memberships on Novell eDirectory

A full functional wizard to manage group memberships is described in this example. The wizard contains a number of projects, to implement the wizard.

Besides the LDAP script actions, the wizard project scripts contain a lot of other UMRA actions that are used to make the wizard more user-friendly. The wizard can be extended in many ways to improve the functionality.

The wizard contains 2 screens: The first screen is used to select a user account.

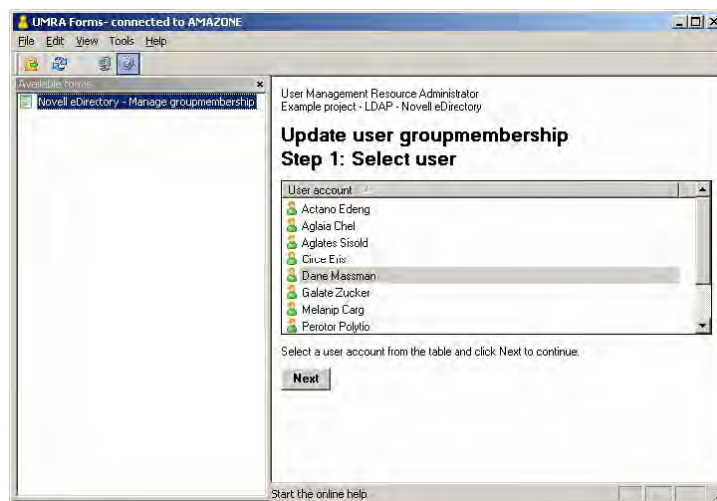


Figure 40: Update user group membership wizard, step 1: Select user.

The second screen shows the available groups and the groups of which the user account is a member. The end-user can add the selected user account to the available groups and remove the group membership from groups of which the user account is a member.



Figure 41: Update user group membership wizard, step 2: Select group.

The wizard contains a number of projects. The projects can be found at the following location, relative to the UMRA Console program directory:

.\Example projects\LDAP\Novell\ManageGroupMembershipWizard

The following table briefly describes the projects of the wizard.

Project	Description
Novell eDirectory - InitializeVars.ufp	Initialize the variables that user used in the other project. This project is executed by the other projects.
Novell eDirectory - Get Users.ufp	Find the user accounts from which the group memberships need to be managed. The users will be presented in a list.
Novell eDirectory - Find Group Name.ufp	From a distinguished group name, find the more user-friendly common name.

Novell eDirectory - Manage groupmembership.ufp	Show a list with user accounts and let the end-user select one of the accounts.
Novell eDirectory - Update Groupmembership.ufp	Show the list with available groups and the groups of which the user account is a member. When one of the buttons is pressed, add the selected user to the selected group or remove the user from a selected group or let the end-user select another user account.

*Table 8: Projects of UMRA application to manage Novell eDirectory group memberships.*



---

### 3.6.6. Linux OpenLDAP

This section describes how user accounts in Linux OpenLDAP can be managed with User Management Resource Administrator (UMRA).

#### Introduction

In many networks environment, Linux servers are integrated in the Active Directory network. For most Linux distributions, an LDAP implementation is available: OpenLDAP. OpenLDAP is an Open Source implementation of LDAP. On Linux, OpenLDAP is used to setup a directory service for different applications and implementations of for instance

- Linux Pluggable Authentication Modules (PAM)
- Linux Name Service Switch (NSS)
- Samba
- FTP/HTTP
- FreeRadius

Such applications can be **LDAP enabled** using OpenLDAP. The UMRA LDAP actions can be used to manage the OpenLDAP directory service in order to create, manage, delete, edit and search directory service items.

Depending on the package and compilation, OpenLDAP supports SSL. In this case, the LDAP communication between the LDAP Client (UMRA) and the LDAP Server (Linux OpenLDAP) is secure using SSL.

In this document, the following Linux and OpenLDAP environment is used:

- Debian GNU/Linux 3.1 (kernel 2.4.27-2-386)
- OpenLDAP, version 2.2.23-8

This section on Linux OpenLDAP covers the following topics:

1. *Setting up a secure Linux OpenLDAP environment* on page 57
2. *Example project to create directory service items* on page 65

#### Secure Linux OpenLDAP environment

To setup a secure Linux OpenLDAP environment, SSL certificates must be installed on the LDAP Server (Linux OpenLDAP) and the LDAP Client (UMRA software).

The OpenLDAP configuration file **slapd.conf** must be updated with the SSL configuration settings. The following parameters must be specified:

Parameter	Description
TLSCipherSuite	Specification of ciphers accepted by the LDAP Server. Examples: RC4:DES:EXPORTS40 HIGH:MEDIUM 3DES:SHA1:+SSL2 See the ciphers(1) manpage distributed with OpenSSL for more information.
TLSCertificateFile	The name of the file that contains the certificate to be used by the LDAP Server
TLSCertificateKeyFile	The name of the file that contains the associated private key of the certificate.

Regarding the certificate, two file names are specified, one for the certificate itself and one for the associated private key. To generate these files, the following procedure can be used.

The certificates are first generated on the Linux computer using the CA.pl script, part of the OpenSSL installation.

```
t4elnx:/ldap-ssl# /usr/lib/ssl/misc/CA.pl -newcert
```

Generating a 1024 bit RSA private key

```
.....++++++
```

```
.....++++++
```

writing new private key to 'newreq.pem'

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

```
-----
```

You are about to be asked to enter information that will be incorporated



into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:nl

State or Province Name (full name) [Some-State]:utrecht

Locality Name (eg, city) []:baarn

Organization Name (eg, company) [Internet Widgits Pty Ltd]:tools4ever

Organizational Unit Name (eg, section) []:development

Common Name (eg, YOUR name) []:t4elnx.tools4ever.local2

Email Address []:

Certificate (and private key) is in newreq.pem

The above listing shows how to create the certificate with the command

**CA.pl -newcert**

The certificate is self signed and no Certification Authority is required.

The contents specified for the fields does not really matter, except for the following fields:

**Common Name:** Specify the dns name of the computer that runs the Linux OpenLDAP.

**Email Address:** Leave this field blank.

When ready, the file **newreq.pem** contains both the private key and the certificate. The private key is password protected. The total file looks like this:

-----BEGIN RSA PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: DES-EDE3-CBC,D704DED67B9622AB

1aUi3gvkxF+kfnpuc0BH7ITU+du4TgoPu/QDMGVUnhuEBN3EXu+m0bIfEWrljqzw  
fujUUNlEmHGO3fKbUaJa7Q5EhWAMWLV7nE/U+ud4Smul6zjXj0Snv6aM6jOvAH  
/9

MHRFO8jB0O1zfmzA6h6wq0v+0GknS1sSH+bLlm1Hb9wGiLRZTopPZUfd1FhTdO  
F

odNWfhVIL2CollnT/+0qHKl1YqF5PCdkKxGLbMC9IM30mZuOZSbDeDQMioTRPQ  
nD

WMgJuWChtHWTcVfriRbEPEimPQ7zOhq5PFsSZXwB8TjXCL8m42knL9h/csBZLjW  
I

Eq4fgCy4odSoQA6bVsRdXHMWYzKLTArUKXkh9yCKimx2EeDVWgl80hm3htus5V  
rR

VCbflBmuA3gghgEFjsrYps5jSsYCIvbesOelyT/K6uafKnax1JsfdKfYKzbMwfOa

Qcq13Mv1EFMlyFROUMMvFiVMjUQnfsaDCMglJxj+XuDFmOWHUUG6CJp0f+XH  
2Sbg

xuACcyMomKIWHzBIGCk6W0p5Xeavnboj8ZiYPcAvQ0vUEGt5owXwJVbyblafuRd  
p

JoHOpyin+q+2pK4oZpfZO0yuTfFP+sLF6iluG77b5QRZS2kLy6mK+8R0qfVjI7Uv

VAltadLhyKKAzeTQLogoArmNe6iAXiJ03cJnVR+qkoW6bmBSuz7fhYD2k8Xyh/hk

9Uh35ALf+GSZ8c5kYVGgLcrrOd7m82bKfGP2fmx3CxWL7wlwSAMP8ZZxNof3vJAf

rr96ju7/0MMjVskyh6XeIXCIDUzbWke+9MVwGsUGnTaxoCN/s1kag==

-----END RSA PRIVATE KEY-----

-----BEGIN CERTIFICATE-----

MIIDXDCCAsWgAwIBAgIJALbVQcGOAzn4MA0GCSqGSIb3DQEBAUAMH0xCzAJ  
BgNV

BAYTAm5sMRAwDgYDVQQIEwd1dHJIY2h0MQ4wDAYDVQQHEwViYWYybjETMB  
EGA1UE

ChMKdG9vbHM0ZXZlcjEUMBIGA1UECXMZLG9VZWxvcG1lbnQxITAFBgNVBAMTG  
HQ0

ZWxueC50b29sczRldmVyLmxvY2FsMjAeFw0wNTEyMDIxMTA4NTRaFw0wNjEyMDIx

MTA4NTRaMH0xCzAJBgNVBAYTAm5sMRAwDgYDVQQIEwd1dHJlY2h0MQ4wDAYDVQQH

EwViYWYybJETMBEGA1UEChMKdG9vbHM0ZXZlcjEUMBIGA1UECXMZGV2ZWxvY2F1

bnQxITAFBgNVBAMTGHQ0ZWxueC50b29sczRldmVyLmxvY2FsMjCBnzANBgkqhkiG

9w0BAQEFAAOBjQAwYkCgYEAq6flBA9IsTX3dUwN5pNIGM3RTE4CtnC5HgyLmoNM

LyDLrNLIjSlf717aNCae1RzpLZnezHiug7dRZKlcqBjGp1wmTohoIbSiHJSOdKp

B5YK4nT2oRyrGnFM/XtftagosOQnWOYCEk3iA5lyk28i4wMZpl6Ad//oZEDBg47C

WHMCAwEAAaOB4zCB4DADBgvNVHQ4EFgQUOYKI1q4QzIHILBVLWpCikwlvhWAwgbAG

A1UdlwSBqDCBpYAUOYKI1q4QzIHILBVLWpCikwlvhWChgYGkfzB9MQswCQYDVQQG

EwJubDEQMA4GA1UECBMHdXRyZWNoDEOMAwGA1UEBxMFYmFhcm4xEzARBgNVBAoT

CnRvb2xzNGV2ZXlxFDASBgNVBA5TC2RldmVsb3BtZW50MSEwHwYDVQQDEXh0NGVs

bngudG9vbHM0ZXZlcj5sb2NhbkDCCQC21UHBjgM5+DAMBgNVHRMEBTADAQH/MA0G

CSqGSib3DQEBBAUAA4GBAGqhYqMj6p1h6zoF/uTIXUho9aIKYeFmggwr7mm4PXJV

4KDYWD/XPNIHEJxOj0Y9zOJmsTIN+/pYBLm6xYri5Lbm9NWS3AmM0Gpn63LD8MB

O1CqEFOMWot4GSBHGkkJF/9WOkQHCFunS3t7bYQyhCM1QdfsWI52Z77FAcYjrGHe

-----END CERTIFICATE-----

To remove the password protection from the private key and to export the private key that is used by the LDAP Server, enter the following command:

**openssl rsa -in newreq.pem slapd-key.pem**

On output, the file **slapd-key.pem** contains the private key with no password protection.

-----BEGIN RSA PRIVATE KEY-----

```
MIICXQIBAAKBgQCrp+UED0ixNfd1TA3mk0gYzdFMTgK2dzkeDluag0wvIMus0siK
NKV/vXto0Jp7VHOktmd7MeK6Dt1FkohyoGManXCZOiGghtKlcll50qkHlgridPah
HKsacUz9e1+1qCiw5CdY5glSTelDkjKTbyLjAxmmXoB3/+hkQMGDjsJYcwlDAQAB
AoGBAJ/IQg/5CLaB1aM+mAg7EOJ/ncGdPSuofNz/xJ7GRRX1T6QJqGIMzkjiQO2O
uwe80AgTHOuFuXOk2vqul0InG0gt561TgpYn8NA987MGYMsj5Vw/wV+bl+tZW/
9p
ZoFJlRrdlxtfrOsejGlpxCGs+TWdzzuecoqIY7nhZSr9CTiRAKEA047LiBn0mEym
leQv6a3UXw23VvxGwkdAD9OQM9YZWI7lycXdKQPL3VYbYMUq0v9MEGJk+zGr4
eYu
EQS7iT3TGQJBAM+3PifZwz7No/hmkfjELNNB23C3kwQCpNy9knHWbrMEeJQOF
ucK
SC+1b2/D+RZ55+2zeJnLC9zdqg1WiLc8pWsCQDIFuRf5XtW0NAz0H3x1kL7C6dV/
k
qotB2rfuIGXIGkj6096R8FOAMZqUCwIhlzxT3PW6jXfrdlrNU79LtrFqyVECCQDI
J0vWfKj+Klv7PWMIcmu7OfepWstojt+r8WRfG4DaMdG64QTCpw6+lJf6W733IYsS
auEoWRbaQiKt7ZeZ8e93AkBLRx6O3ez3Jj/5hDL57jXFeg/THV59qCEBOKcKjPA7
BAnnjPnQGK5h32g4IfU5Mf0jQTapxu1icNhstFhwFAIq
```

-----END RSA PRIVATE KEY-----

In a real environment, this file should be highly protected since it contains the main secret: the private key (You should never publish the contents of this file in a document). From the other file, **newreq.pem**, you need to create a file that contains the certificate only. In this example description, the certificate from the file is stored in a new file **slapd-cert.pem**.

This file should contains something like this:

-----BEGIN CERTIFICATE-----

MIIDXCCAsWgAwIBAgIJALbVQcGOAzn4MA0GCSqGSIb3DQEBAUAMH0xCzAJBgNV

BAYTAm5sMRAwDgYDVQQIEwd1dHJlY2h0MQ4wDAYDVQQHEwViYWYybjETMBEGA1UE

ChMKdG9vbHM0ZXZlcjEUMBIGA1UECXMZGV2ZWxvcG1lbnQxITAfBgNVBAMTG

ZWxueC50b29sczRldmVyLmxvY2FsMjAeFw0wNTEyMDIxMTA4NTRaFw0wNjEyMDIx

MTA4NTRaMH0xCzAJBgNVBAYTAm5sMRAwDgYDVQQIEwd1dHJlY2h0MQ4wDAYDVQQH

EwViYWYybjETMBEGA1UEChMKdG9vbHM0ZXZlcjEUMBIGA1UECXMZGV2ZWxvcG1l

bnQxITAfBgNVBAMTGHQ0ZWxueC50b29sczRldmVyLmxvY2FsMjCBnzANBgkqhkiG

9w0BAQEFAAOBjQAwgYkCgYEAq6fIBA9IsTX3dUwN5pNIGM3RTE4CtnC5HgyLmoNM

LyDLrNLIjSlf717aNCae1RzpLZnezHiug7dRZKlcqBjGp1wmTohoIbSiHJSOdKp

B5YK4nT2oRyrGnFM/XtftagosOQnWOYCEk3iA5lyk28i4wMZpl6Ad//oZEDBg47C

WHMCAwEAAaOB4zCB4DAdBgNVHQ4EFgQUOYKI1q4QzIHILBVLWpCikwlvhWA

A1UdlwSBqDCBpYAUOYKI1q4QzIHILBVLWpCikwlvhWChgYGkfzB9MQswCQYDVQQG

EwJubDEQMA4GA1UECBMHdXRyZWNoDEOMAwGA1UEBxMFYmFhcm4xEzARBgNVBAoT

CnRvb2xzNGV2ZXlxFDASBgNVBA5TC2RldmVsb3BtZW50MSEwHwYDVQQDEEx0NGVs

bngudG9vbHM0ZXZlcjE5sb2NhbkDCCQC21UHBjgM5+DAMBgNVHRMEBTADAQH/MA0G

CSqGSIb3DQEBAUAA4GBAGqhYqMj6p1h6zoF/uTIXUho9aIKYeFmggwr7mm4PXJV

4KDYWD/XPNIHEJxOj0Y9zOJmsTIN+/pYBLm6xYri5Lbm9NWS3AmM0Gpn63LDb8MB

O1CqEFOMWot4GSBHGkkJF/9WOkQHCfunS3t7bYQyhCM1QdfsWI52Z77FAcYjr  
GHe

-----END CERTIFICATE-----

Now, update OpenLDAP configuration file, so that it contains the following lines to enable SSL:

```
TLSCipherSuite      HIGH:MEDIUM
TLSCertificateFile   /ldap-ssl/slapd-cert.pem
TLSCertificateKeyFile /ldap-ssl/slapd-key.pem
```

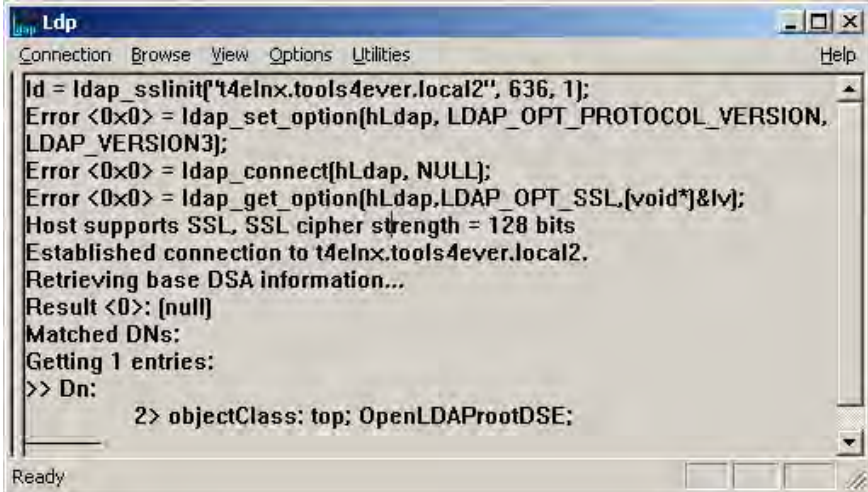
The file names should point to the locations of the files with the certificate and the associated private key. Finally, restart the LDAP Server:

```
/etc/init.d/slapd restart
```

The LDAP Server is now able to communicate using SSL. Now, the certificate must be imported on the computer that runs the UMRA software: Copy the file **slapd-cert.pem** to the computer that runs the UMRA software and follow the instructions as described in section:

**Import the certificate on the UMRA computer.**

When ready, the test with LDP.EXE, part of the Windows Support Tools, should show a result as in the following figure:



```

Ldp
Connection  Browse  View  Options  Utilities  Help
ld = ldap_sslinit("t4elnx.tools4ever.local2", 636, 1);
Error <0x0> = ldap_set_option(hLdap, LDAP_OPT_PROTOCOL_VERSION,
LDAP_VERSION3);
Error <0x0> = ldap_connect(hLdap, NULL);
Error <0x0> = ldap_get_option(hLdap,LDAP_OPT_SSL,(void*)&lv);
Host supports SSL, SSL cipher strength = 128 bits
Established connection to t4elnx.tools4ever.local2.
Retrieving base DSA information...
Result <0>: [null]
Matched DN's:
Getting 1 entries:
>> Dn:
      2> objectClass: top; OpenLDAProotDSE:
Ready

```

By default, you can then bind with the **admin** account:

**cn=admin,dc=tools4ever,dc=local2**

to authenticate the user account.

### Creating directory service items with OpenLDAP on Linux

The example project describes a mass project to import directory service items. On Linux, different applications use different LDAP schemas. In this example project, the default general schema is used. To create directory service items that are used by LDAP enabled applications, modification might be required. Examples of these applications are Samba, PAM, NSS, FTP/HTTP, FreeRadius.

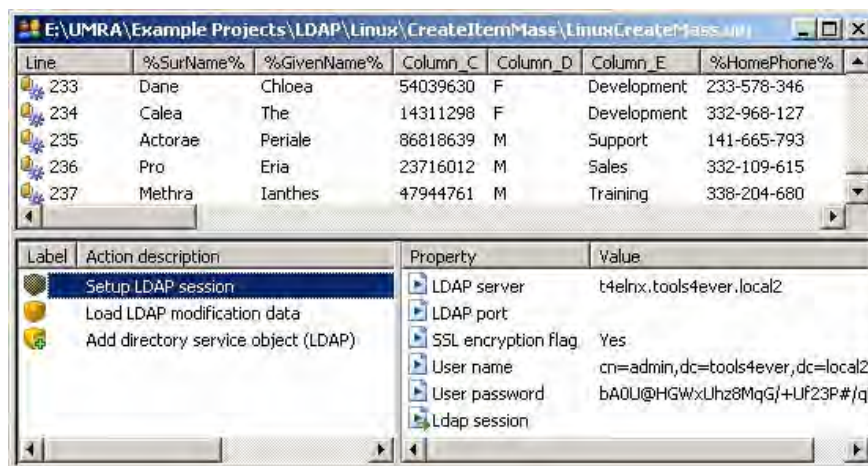
The example project can be found in the following location, relative to the UMRA Console program directory:

.\\Example Projects\\LDAP\\Linux\\CreateItemMass\\LinuxCreateMass.upj

The project contains embedded input data representing user accounts. For each line of the input data, the project repeats the following steps:

1. *Setup a secure LDAP session with the LDAP Server running on the Linux computer on page 66;*
2. *Setup an LDAP modification data structure that contains all the attributes to create a person directory service item on page 67.*

3. Add the item to the directory service in a specific organizational unit on page 68.



The following sections describes the script of the project in detail.

#### *Setting up a secure session with Linux LDAP Server*

The LDAP session is setup with the Linux computer **t4elnx.tools4ever.local2**. To authenticate the session, the full distinguished name of the administrator root account is specified. The password of the account is encrypted. It is decrypted just before the UMRA script engine needs it to access the LDAP Server.

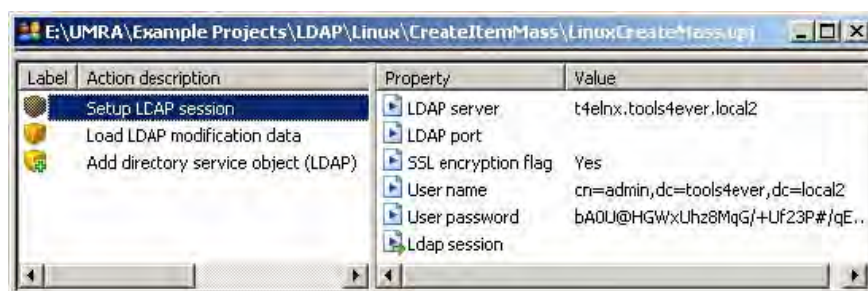


Figure 42: Script action to setup a secure LDAP session with the OpenLDAP Linux server.



The session is encrypted using SSL. When the action is executed, the LDAP session is initialized. The associated session data is stored in the variable specified for property **Ldap session** (default variable name: %LdapSession%). The variable is used in the subsequent script actions.

#### *Specifying LDAP attributes and values of directory service item*

In the next script action, the LDAP modification data structure is initialized. The structure must contain all the attributes and values of the new directory service item.

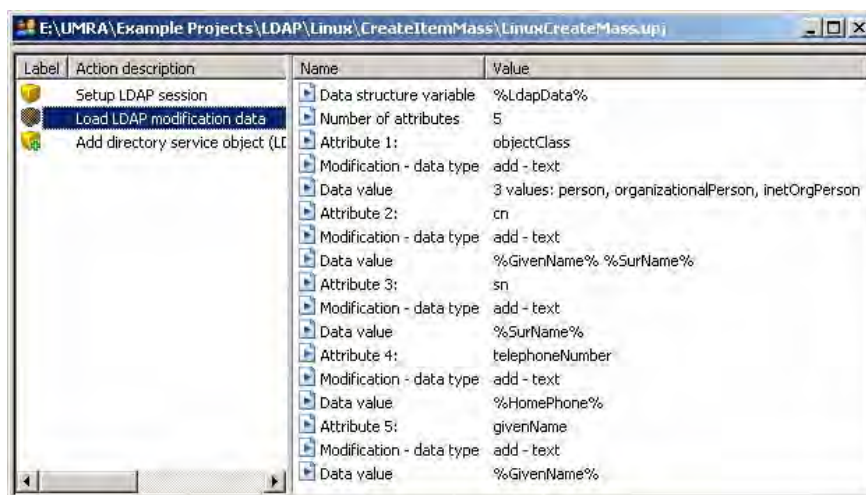


Figure 43: Script action to initialize the LDAP modification data that is used to create a person item in the directory service on Linux using OpenLDAP.

To create a person in the directory service, the following attributes are used:

Attribute	Description
objectClass	Defines the type of the directory service item that must be created. Contains multiple values: inetOrgPerson, organizationalPerson, person. The attribute values are the same for items created.

cn	The common name of the person. SAM account name of the new user account. The value is set equal to the variable combination <b>%GivenName% %SurName%</b> that is read from the input file.
sn	The surname of the new person. This is a mandatory attribute. The value is set equal to the variable <b>%SurName%</b> .
telephoneNumber	An example of an attribute as defined in the schema. The value is set equal to the variable <b>%HomePhone%</b> , copied from the input file.
givenName	An example of an attribute that is available though one of the parent object classes: The <b>givenName</b> attribute is defined for object class <b>inetOrgPerson</b> . Since the <b>objectClass</b> values for the new directory service item include the value <b>inetOrgPerson</b> , the attribute exists for the object that is created with this action.

*Table 9: LDAP attributes to create an user account in Active Directory.*

#### *Adding the person directory service item*

Finally, the person directory service item is created with this action.



*Figure 44: Script action to add the person to the Linux OpenLDAP directory service.*

#### **Add person directory service item**

The **LDAP session** and **Object data** properties are specified using variables. The **New object name** property specifies the full object distinguished name of the item and thus determines the position in the directory service tree. This name must be unique within the directory service. In this example, all person items are created in the same organizational unit: **people**. The common name (**cn**) is copied from the input data:

```
cn=%GivenName% %SurName% ,  
ou=people,dc=tools4ever,dc=local2
```

Note that the common name (**cn**) is specified on two locations:

1. As an attribute in action **Load LDAP modification data**.
2. In this action **Add directory service object (LDAP)**.

If the names do not correspond, a naming violation error occurs and the person directory service item is not created.

### Logging information

UMRA Console log of user creation using secure LDAP

When executed successfully, the UMRA Console log file produces a log file as shown below:

```
Starting User Management Resource Administrator session, build 1213 at  
13:45:12 12/06/2005
```

```
13:45:12 12/06/2005 ***** Processing entry 66...
```

```
13:45:12 12/06/2005 Variable 1: %GivenName%=Tece
```

```
13:45:12 12/06/2005 Variable 2: %SurName%=Cowsake
```

```
13:45:12 12/06/2005 Variable 3: %HomePhone%=304-411-583
```

```
13:45:12 12/06/2005 Variable 4: %NowDay%=06
```

```
13:45:12 12/06/2005 Variable 5: %NowMonth%=12
```

```
13:45:12 12/06/2005 Variable 6: %NowYear%=2005
```

```
13:45:12 12/06/2005 Variable 7: %NowHour%=13
```

```
13:45:12 12/06/2005 Variable 8: %NowMinute%=45
```

```
13:45:12 12/06/2005 Variable 9: %NowSecond%=12
```

```
13:45:12 12/06/2005 Setting up LDAP sessions with host  
't4elnx.tools4ever.local2'. Using SSL encryption: 'Yes'.
```

```
13:45:12 12/06/2005 User name: 'cn=admin,dc=tools4ever,dc=local2'.
```

```
13:45:12 12/06/2005 Secure LDAP session established with host  
't4elnx.tools4ever.local2' (Protocol: 'TLS 1.0 client-side', encryption: 'RC4
```

stream', cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA', 1024 bits).

13:45:12 12/06/2005 Authenticating user 'cn=admin,dc=tools4ever,dc=local2'...

13:45:12 12/06/2005 User 'cn=admin,dc=tools4ever,dc=local2' successfully authenticated on LDAP server host 't4elnx.tools4ever.local2'.

13:45:12 12/06/2005 LDAP session information stored in variable '%LdapSession%'.

13:45:12 12/06/2005 Storing LDAP modification data in variable '%LdapData%'.

13:45:12 12/06/2005 LDAP modification data:

13:45:12 12/06/2005 \*\*\*\*\* Modification data element: 0  
\*\*\*\*\*

13:45:12 12/06/2005 Operation: 'add', type of data: 'text'

13:45:12 12/06/2005 Attribute: 'objectClass'

13:45:12 12/06/2005 Value 0: 'person'

13:45:12 12/06/2005 Value 1: 'organizationalPerson'

13:45:12 12/06/2005 Value 2: 'inetOrgPerson'

13:45:12 12/06/2005 \*\*\*\*\* Modification data element: 1  
\*\*\*\*\*

13:45:12 12/06/2005 Operation: 'add', type of data: 'text'

13:45:12 12/06/2005 Attribute: 'cn'

13:45:12 12/06/2005 Value 0: 'Tece Cowsake'

13:45:12 12/06/2005 \*\*\*\*\* Modification data element: 2  
\*\*\*\*\*

13:45:12 12/06/2005 Operation: 'add', type of data: 'text'

13:45:12 12/06/2005 Attribute: 'sn'

13:45:12 12/06/2005 Value 0: 'Cowsake'

13:45:12 12/06/2005 \*\*\*\*\* Modification data element: 3  
\*\*\*\*\*

13:45:12 12/06/2005 Operation: 'add', type of data: 'text'

13:45:12 12/06/2005 Attribute: 'telephoneNumber'

13:45:12 12/06/2005 Value 0: '304-411-583'

13:45:12 12/06/2005 \*\*\*\*\* Modification data element: 4  
\*\*\*\*\*

13:45:12 12/06/2005 Operation: 'add', type of data: 'text'

13:45:12 12/06/2005 Attribute: 'givenName'

13:45:12 12/06/2005 Value 0: 'Tece'

13:45:12 12/06/2005 Adding LDAP directory service object 'cn=Tece Cowsake, ou=people,dc=tools4ever,dc=local2' with LDAP modification data obtained from variable '%LdapData%'.

13:45:12 12/06/2005 LDAP directory service object 'cn=Tece Cowsake, ou=people,dc=tools4ever,dc=local2' successfully added.

13:45:12 12/06/2005 \*\*\*\*\* Ready processing entry 66...

13:45:12 12/06/2005 Total number of script action execution errors: 0.

End of session

The log file shows the following topics:

1. initialization of the secure LDAP session;
2. the authentication of the connecting account;
3. initialization of the LDAP modification data
4. creation of the directory service item



---

### 3.6.7. Microsoft Active Directory

This section describes how Microsoft Active Directory objects can be managed using UMRA LDAP actions.

#### Introduction

#### Native UMRA actions

For a Microsoft Active Directory environment, UMRA contains many native actions. It is advised to use these (AD) actions to manage Active Directory. However, since Active Directory does also support LDAP, it is also possible to manage Active Directory objects using the UMRA LDAP actions. How to do that is described in this section.

#### Deployment scenario

In case no trust relation exists between the computer that runs the UMRA software and the Active Directory domain controllers, the LDAP actions can be used to manage Active Directory. In such an environment, the native UMRA actions cannot be used. Because of security reasons, such an environment should be configured using encrypted communication only.

This section describes how to:

- *setup a secure Active Directory Windows 2003 LDAP environment on page 73.*

and gives some example projects to:

- *create a user account on page 85;*
- *reset passwords on page 93;*
- *update group memberships on page 115.*

#### Secure LDAP Active Directory environment

By default, the Microsoft LDAP implementation does not support secure LDAP. To setup secure LDAP using SSL, certificates must be installed on both sides, the LDAP Server and LDAP Client. In this case, the LDAP Server is the domain controller running Active Directory. The LDAP Client is the UMRA software, either the UMRA Console application or the UMRA Service.

The certificates required to run secure LDAP using SSL can be configured in many ways. The concept is always the same:

1. The Active Directory domain controller uses a special certificate that is issued by a trusted certification authority.
2. The UMRA software (computer) trusts the certification authority that issues the certificate to the Active Directory domain controller.

Creating the certificate listed in step 1 requires a special procedure, as described in article Q321051. In this document, the same steps are used and described. Also, the procedure to setup a Certification Authority is described.

First, a certificate request is created. Next, a Certification Authority (CA) is setup and the certificate is signed, e.g. issued by the certification authority. Finally, the root certificate of the certification authority is exported and then imported by the computer that runs the UMRA software.

In this procedure the environment used runs Active Directory on Windows 2003 Standard Edition. For Windows 2000, a similar procedure can be used. The Certification Authority is installed on a Windows 2003 domain controller. For other versions, the procedure might be different.

#### **Creating an Active Directory domain controller certificate request**

Log on to domain controller

This topic follows the guidelines of article Q321051. Log on to the domain controller (LDAP Server) with an enterprise administrator account.

To create the certificate request, the **certreq.exe** program is used. The **certreq.exe** program is part of the Windows installation and requires a text input file to generate a certificate request.

With your favorite ASCII editor (notepad.exe?), create a file with the following contents:

[Version]

Signature="\$Windows NT\$"

[NewRequest]

Subject = "CN=king.tools4ever.local3"



; replace with the FQDN of the DC

KeySpec = 1

KeyLength = 2048

; Can be 1024, 2048, 4096, 8192, or 16384.

; Larger key sizes are more secure, but have

; a greater impact on performance.

Exportable = TRUE

MachineKeySet = TRUE

SMIME = False

PrivateKeyArchive = FALSE

UserProtected = FALSE

UseExistingKeySet = FALSE

ProviderName = "Microsoft RSA SChannel Cryptographic Provider"

ProviderType = 12

RequestType = PKCS10

KeyUsage = 0xa0

[EnhancedKeyUsageExtension]

OID=1.3.6.1.5.5.7.3.1

In the file, the entry regarding the subject,

**Subject = "CN=king.tools4ever.local3"**

must be changed to contain the fully qualified domain name of the Active Directory domain controller that is going to support secure LDAP.  
Example:

**Subject = "CN=OtherServer.mydomain.com"**

Save the file to **ldapcert.inf**. From a command prompt, create the request file with **certreq.exe**:

**certreq -new ldapcert.inf ldapcert.req**

A new file is now created: **ldapcert.req**. This is the base64 encoded request file and it contains something like this:

-----BEGIN NEW CERTIFICATE REQUEST-----

MIIELDCCAxQCAQAwITefMB0GA1UEAxMWa2luZy50b29sczRldmVyLmxvY2FsMzCC

ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANFwryRM0qxBNQKr/fQIZr  
bL

gqs9LMWFSolAzVA342N2RisBLXFtuoNxZPkD0UIQmcLLjwBA8svmVsfLRMa+0yg

GKnxYkrpVLOwGkEsLtPKKrt/ZfS1leChkTSC7xZ2U/ajx0qVqUyxtEfGvNI9t7gO

Qr5o0f4YdeId70Y42J2uxmYophZQQRwfdXdE8RB98TjXm+ATdVbKw500Egv7oYD  
9

E5eH7tk3BVNzL65n+MdUTl3jtg7LiivFBbZrDy4WbDjQDcBTx8T98E6sgtOt5iMU

W3rdpPtg8kPWwCDCFPaXTeaRnGWx5QlrfanoOml/EhxclXi82vCAH6HkTzy8rU  
C

AwEAAaCCAcQwGgYKKwYBBAGCNw0CAzEMFgo1LjluMzc5MC4yMFAGCSqGS1b3  
DQEJ

DjFDMEewHQYDVR0OBByEFHf9nSUn4NT5wX9p4jl2tcwHS/2eMBMGA1UdJQQQ  
MMAoG

CCsGAQUFBwMBMA5GA1UdDwQEAwIFoDBUBgkrBgEEAYI3FRQxRzBFAGEBDB9j  
b3Vu

dC5ub2JpbGl0eS50b29sczRldmVyLmxvY2FsDBZOT0JTTElUWVxBZG1pbmlzdHJh

dG9yDAdjZXJ0cmVxMIH9BgorBgEEAYl3DQICMYHuMIHrAgEBHloATQBpAGMAcgbv

AHMAbwBmAHQAIABSAFMAQQAgAFMAQwBoAGEAbgBuAGUAbAAgAEMAcbB  
5AHAAAdABv

AGcAcgBhAHAAaABpAGMAIABQAHIAbwB2AGkAZABIAHIDgYkAAAAAAAAAAAAA  
AAAAA

[illegible][illegible]

AAADANBgkqhkiG9w0BAQUFA  
AOCAQEA

nTAOKjTTbz/ABAH CZRNmn/SSj5w7DoMBUP07l8QQMf4ruI0ClEuX5jhlm+jwnypY

pDNHnn2uRI08hN5jwOcc/36DGNASgu9cOg3s/FCHnDkhMotqST4UgjH8bVBxfTr  
P

ryAswB4CtFDPK4Po9+Fz/Tenb1rD4yC0hvYL2m+Gwyl9rupfj9eyy7VaFZDeHlR

2DkGjF7fOiwjZgXi7jy4w0GtC53hWYWxfTaRTPjKuoGFlwDcUHNucdSEQ216xTg7

yLgyyQv8imBI98dr+XXVJeAQk/ByD8uCU0DWM2M64i2ccw8QhlyOPyj36914K2z  
d

WJSRbwjM1KpvtYFrzwCGJg==

-----END NEW CERTIFICATE REQUEST-----

### Creating a Certification Authority

Log on to computer that runs Certification Authority

In this example, **Certification Authority** is installed on another domain controller running Windows 2003. For other configurations, you are referred to Microsoft documentation for more information on how to setup a Certification Authority.

Log on to the computer with **enterprise administrator access rights**.  
Select **Start, Control Panel, Add or Remove programs**. Click the button  
**Add/Remove windows components**. Select entry **Certificate Services**

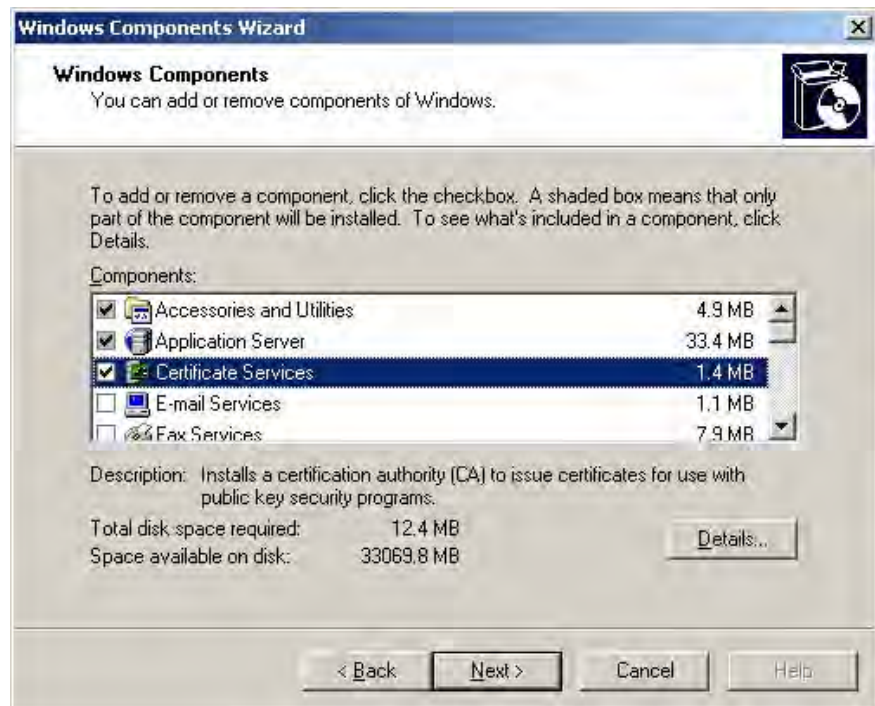


Figure 45: Installation of Certificates Services on a Windows 2003 domain controller.

Click **Next**. When asked, select the option to create a **Stand-alone root CA**.

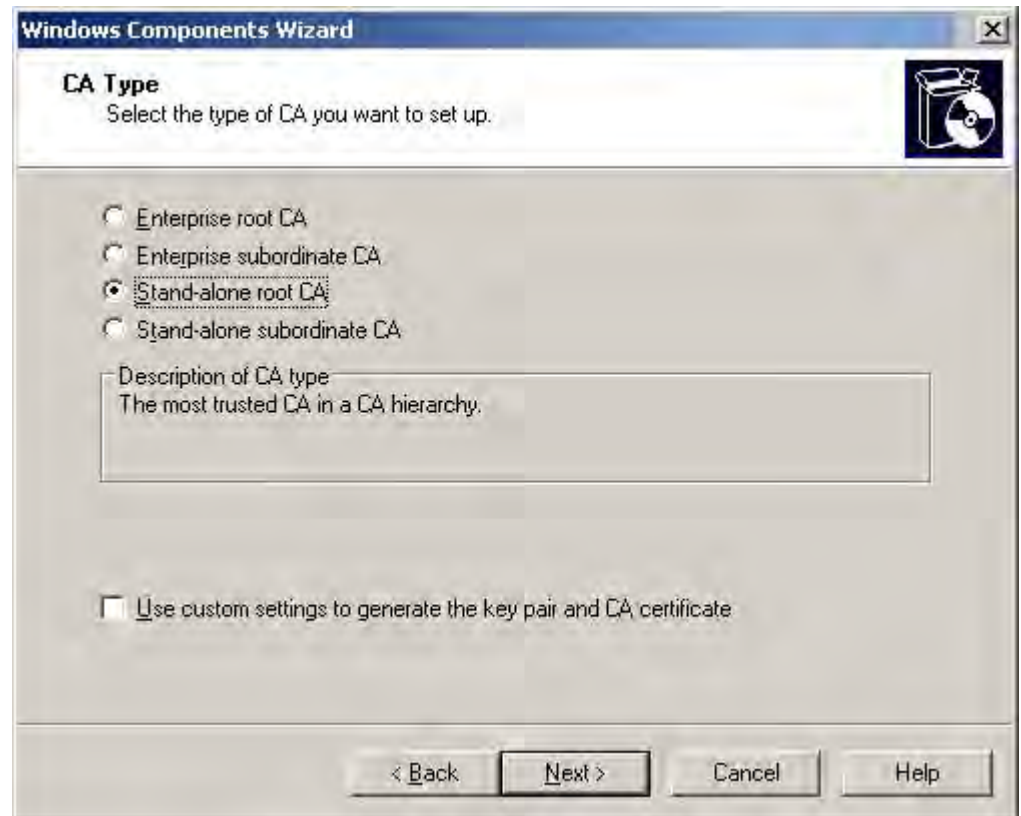


Figure 46: Selection of Certification Authority: Stand-alone root CA.

Follow the wizard instructions and specify the name of the Certification Authority.

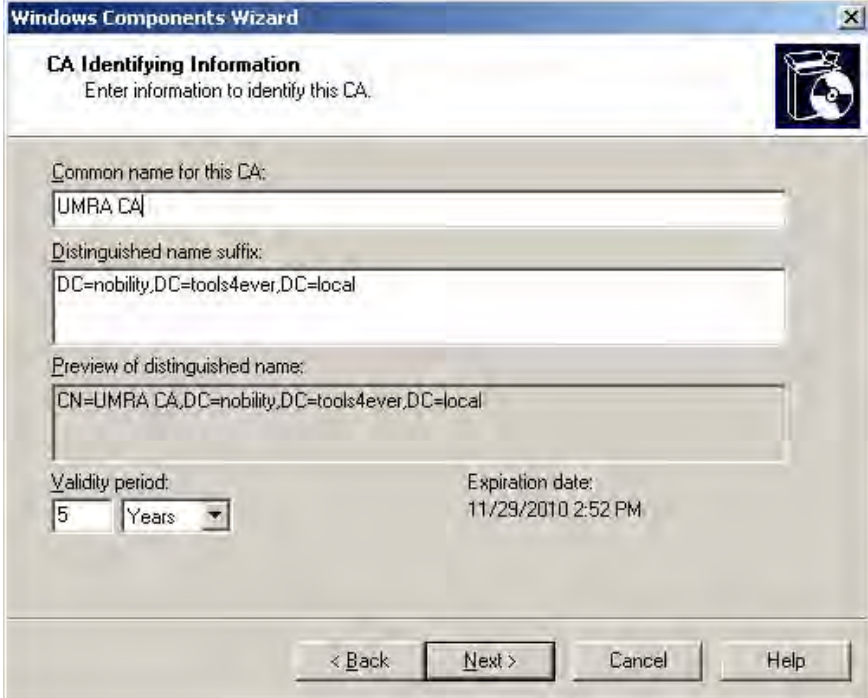
The image shows a screenshot of the 'Windows Components Wizard' window, specifically the 'CA Identifying Information' step. The window has a blue title bar with the text 'Windows Components Wizard' and a close button. Below the title bar, the step name 'CA Identifying Information' is displayed, followed by the instruction 'Enter information to identify this CA.' and a CD icon. The main area contains several input fields: 'Common name for this CA:' with the text 'UMRA CA', 'Distinguished name suffix:' with the text 'DC=nobility,DC=tools4ever,DC=local', and 'Preview of distinguished name:' with the text 'CN=UMRA CA,DC=nobility,DC=tools4ever,DC=local'. At the bottom left, there is a 'Validity period:' section with a dropdown set to '5' and a 'Years' label. To the right, the 'Expiration date:' is shown as '11/29/2010 2:52 PM'. At the bottom of the window are four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Figure 47: Specification of the Certification Authority identification information.

Follow the instructions of the wizard. When finished, the certification authority is installed.

#### Sign the certificate request by the Certification Authority

In this step, the Certification Authority converts the certificate request to a real certificate by **signing/issuing** the request.

Click **Start, All Programs, Administrative Tools, Certification Authority**. The MMC shows the **Certification Authority** snap-in. Select the **Certification Authority** and select menu option **All tasks, Submit new request....**

Browse to the file that contains the certificate request **ldapcert.req** and submit the request. If the computer that runs the LDAP Server and the computer that runs the Certification Authority are not connected, you need to use a diskette or memory stick to access the file.

The request is now processed by the Certification Authority. When ready the request can be selected from the section with **Pending Requests**.



Figure 48: Issue the submitted certification request.

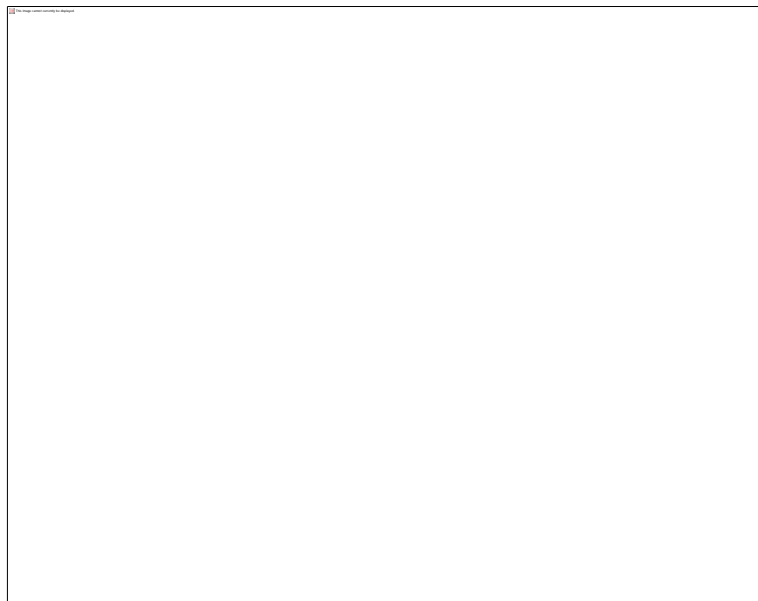
Select menu option **Issue** to accept the request. The certificate is then stored in the section **Issued Certificates**. Select the certificate from the section **Issued Certificates** and select menu option **Open**.



Figure 49: Result certificate, issued by the Certification Authority.

Click on the **Details** tab and select the button **Copy to File...** to export the certificate to a file. Follow the wizard instructions. When asked, select the format **Base-64 encoded binary X.509 (.CER)**. For the name of the file, select **Idapcert.cer**. Complete the wizard.

To export the root certificate, select Certification Authority and select menu option **Properties**.



Click **View Certificate**, select **Details** and click on the button **Copy to File** to export the root certificate of the Certification Authority. For the name of the file, enter **LdapRootCA.cer**.

On the domain controller that runs Active Directory, you need to install both the root certificate of the Certification Authority and the created certificate.

Log on to the domain controller as an enterprise administrator and start the MMC. (**Start, Run, mmc**). Add the **Certificates** snap-in (**File,**



**Add/Remove snap-in**, click **Add** and select **Certificates**). Select the option to manage certificates for the **Computer** account of the **Local Computer**.

Navigate to the certificates item **Trusted Root Certification Authorities**, **Certificates** and select menu option **All Tasks, Import**.

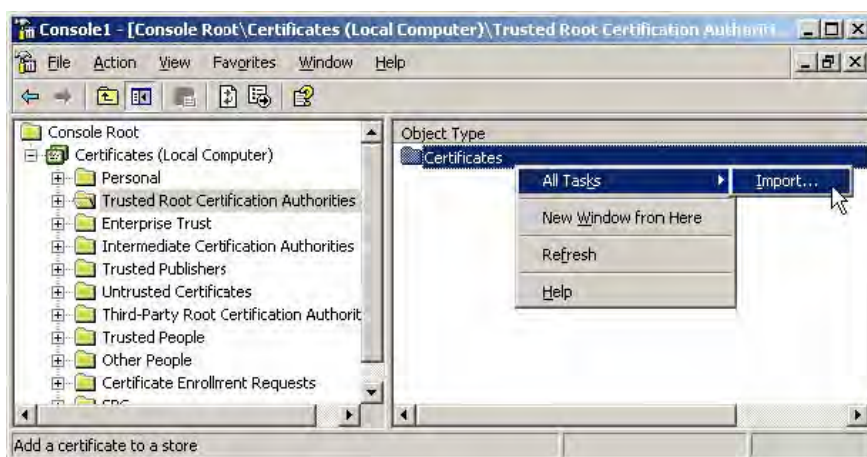


Figure 51: Import the root certificate of the Certification Authority on the Active Directory domain controller.

Follow the wizard instructions and import the root certification file **LdapRootCA.cer**.

When finished, the root certificate of the Certification Authority is installed on the domain controller.

#### Importing the LDAP Server certificate

Finally, on the domain controller that runs Active Directory, you need to accept the certificate signed by the Certification Authority. From a command prompt, navigate to the directory that contains the certificate **ldapcert.cer** and issue the following command:

```
certreq -accept ldapcer.cer
```

The certificate is now installed. To verify the certificate installation, start the MMC and open the snap-in that manages the certificates on the local computer. In the tree, browse to the location **Certificates (Local Computer)**, **Personal**, **Certificates**. A certificate issued to the domain

controller should exist.



Figure 52: Verification of the purpose of the certificate.

Select the certificate and choose menu option **Properties**. The **Certificate purposes** should show **Server Authentication**.

To finish the configuration on the domain controller, restart the domain controller.

#### Setting up the UMRA (LDAP Client) computer

The computer that runs the UMRA software needs to have the root certificate of the Certification Authority installed. To do so, repeat the steps of topic **Import root certificate Certification Authority**, but this time, import the certificate on the computer that runs the UMRA software.

#### Verifying secure LDAPS using SSL

The secure LDAP connection can be tested with UMRA or with the Active Directory Administration Tool **LDP.EXE**, part of the **Windows Support Tools**. (Windows 2003 only)

Start the tools LDP.EXE from a command prompt in the **Windows Support Tools**. Select menu option **Connection, connect**.

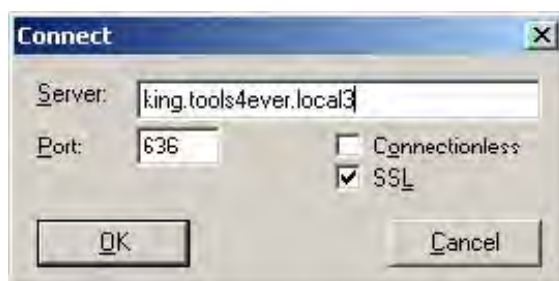


Figure 53: Test the LDAP SSL connection using LDP.EXE from the Windows Support Tools.

Specify the name of the LDAP Server, the default SSL port 636 and check the option SSL.

Press **OK**. When the connection is setup successfully, the window shows the connection information.

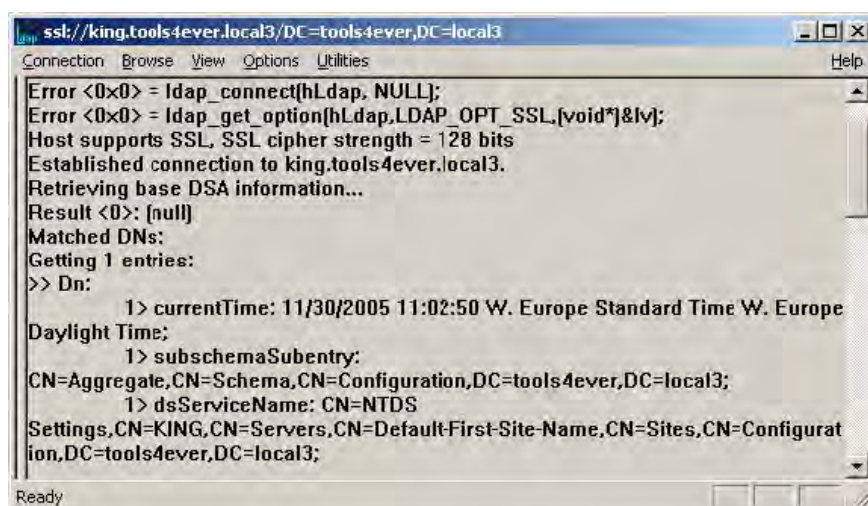


Figure 54: Connection information when a successful connection is established.

### Creating user accounts in Microsoft Active Directory using LDAP

This example describes a mass project to import user accounts into Microsoft Active Directory with UMRA using the UMRA LDAP actions instead of the normal UMRA Active Directory action.

### User secure LDAP when connecting to a not trusted environment

Such a project can be used when a foreign domain is managed and the UMRA software runs on a computer that has no trust relationship with the domain. By using SSL, the application is completely secure.

The example project can be found in the following location, relative to the UMRA Console program directory:

#### .\Example

**Projects\LDAP\ActiveDirectory\AddUserMass\LdapADUserAccountMass.upj**

The project contains embedded fake input data representing user accounts. For each line of the input data, the project repeats the following steps:

1. *Setup a secure session with the LDAP Server running on the Active Directory domain controller on page 87.*
2. *Setup an LDAP modification data structure that contains all the attributes and values required to create a user account on page 88.*
3. *Add the account in the specified container (domain, container or organizational unit) of Active Directory on page 90.*

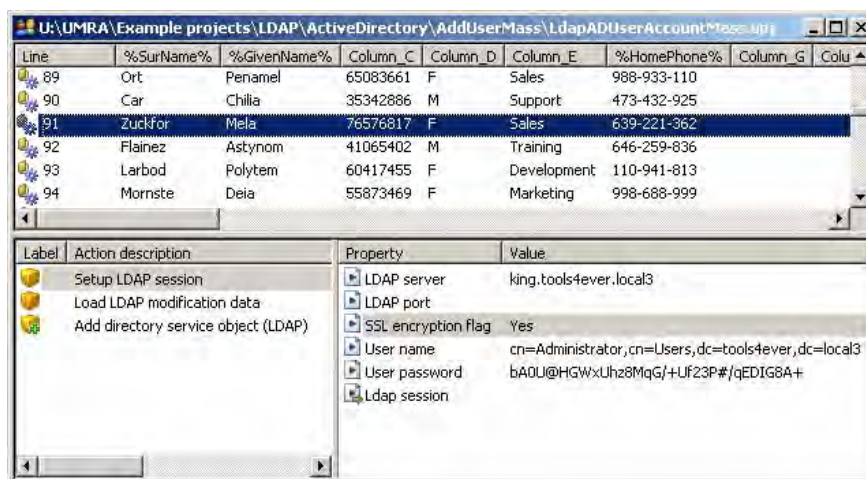


Figure 55: Example project to create mass user accounts in Active Directory using secure LDAP.

The following sections describes the script of the project in detail.

### *Setting up a secure session with Active Directory domain controller*

The LDAP session is setup with the Active Directory domain controller **king.tools4ever.local3**. To authenticate the session, the full distinguished name of an administrator account is specified. The account must have access rights to create the account. The password of the account is encrypted. It is decrypted just before the UMRA script engine needs it to access the LDAP Server.

Note: The password can only be set using SSL.



Figure 56: Script action to setup a secure LDAP session with an Active Directory domain controller.

The session is encrypted using SSL. This is secure and **required in order to (re)set passwords of Active Directory user accounts**. See Microsoft's knowledge base article KB269190 for more information.

When the action is executed, the LDAP session is initialized. The associated session data is stored in the variable specified for property **Ldap session** (default variable name: %LdapSession%). The variable is used in the subsequent script actions.

### Specifying Active Directory LDAP attributes

In the next script action, the LDAP modification data structure is initialized. The structure must contain all the attributes and values of the new directory service item.

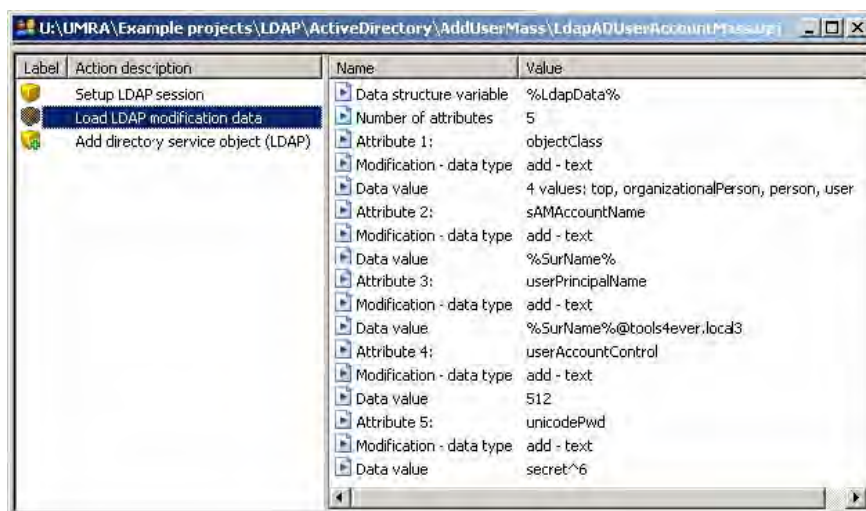


Figure 57: Script action to initialize the LDAP modification data used to create an user account in Active Directory.

To create a user account in Active Directory, the following attributes are used:

Attribute	Description
objectClass	Defines the type of the directory service item that must be created. Contains multiple values: top, organizationalPerson, person, user. The attribute values is the same for all user accounts.
sAMAccountName	The SAM account name of the new user account. The value is set equal to the variable <b>%SurName%</b> that is read from the input file, second column. The SAM account name must be unique in the domain.
userPrincipalName	The official user logon name, specified as <b>%SurName%@tools4ever.local3</b> .
userAccountControl	A number of flags indicating the type of the user account (see below).

unicodePwd	The user account password. This value can only be specified if the LDAP session is secure using SSL. (Note: Internally, this attribute is handled a bit different compared to the other attributes, see knowledge base article KB269190 to check the details)
------------	---

*Table 10: LDAP attributes to create an user account in Active Directory.*

#### User account control flags

To specify the value of the userAccountControl attribute, a simple calculation must be made. Add the values shown in the table below to determine the exact value.

<b>userAccountControl bitmask value</b>	<b>Description</b>	<b>Comment</b>
512	Normal account	Always include this value
2	Disabled account	
4096	Computer account	When included, the account is setup as an computer account
-	User must change password at next logon	This flag cannot be set directory. Instead, the attribute <b>pwdLastSet</b> must be set to 0.
64	User cannot change password	
65536	Password never expires	
32	Password not required	

*Table 11: Overview of the bit flags specified by attribute userAccountControl*

When the user account is created, the Active Directory software checks and updates the value of the **userAccountControl** attribute. When no



password is specified, the flag **User must change password at next logon** is set automatically.

The result of this action is stored in variable **%LdapData%** that is used by the next action.

#### *Adding a directory service object*

Finally, the user account is created with this action.

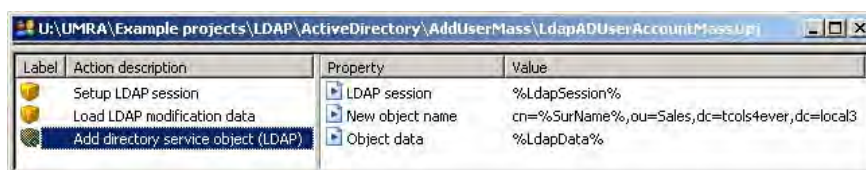


Figure 58: Script action to add the user account to Active Directory.

#### **Adding the user account**

The **LDAP session** and **Object data** properties are specified using variables. The **New object name** property specifies the full object distinguished name of the Active Directory item and thus determines the position in the directory service tree. In this example, all user accounts are created in the same organizational unit. The common name (**CN**) is copied from the input data:

**cn=%SurName%,ou=Sales,dc=tools4ever,dc=local3**

The full name must be unique in the Active Directory tree.

#### **Logging information**

UMRA Console log of user creating using secure LDAP

When executed successfully, the UMRA Console log file produces a log file as shown below:

Starting User Management Resource Administrator session, build 1213 at  
09:29:20 12/02/2005

09:29:20 12/02/2005 \*\*\*\*\* Processing entry 105...

09:29:20 12/02/2005 Variable 1: %GivenName%=Otio



09:29:20 12/02/2005 Variable 2: %SurName%=Methyl

09:29:20 12/02/2005 Variable 3: %HomePhone%=950-491-354

09:29:20 12/02/2005 Variable 4: %NowDay%=02

09:29:20 12/02/2005 Variable 5: %NowMonth%=12

09:29:20 12/02/2005 Variable 6: %NowYear%=2005

09:29:20 12/02/2005 Variable 7: %NowHour%=09

09:29:20 12/02/2005 Variable 8: %NowMinute%=29

09:29:20 12/02/2005 Variable 9: %NowSecond%=20

09:29:20 12/02/2005 Setting up LDAP sessions with host  
'king.tools4ever.local3'. Using SSL encryption: 'Yes'.

09:29:20 12/02/2005 User name:  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'.

09:29:20 12/02/2005 Secure LDAP session established with host  
'king.tools4ever.local3' (Protocol: 'TLS 1.0 client-side', encryption: 'RC4 stream',  
cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA', 2048 bits).

09:29:20 12/02/2005 Authenticating user  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'...

09:29:20 12/02/2005 User 'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'  
successfully authenticated on LDAP server host 'king.tools4ever.local3'.

09:29:20 12/02/2005 LDAP session information stored in variable  
'%LdapSession%'.

09:29:20 12/02/2005 Storing LDAP modification data in variable '%LdapData%'.

09:29:20 12/02/2005 LDAP modification data:

09:29:20 12/02/2005 \*\*\*\*\* Modification data element: 0  
\*\*\*\*\*

09:29:20 12/02/2005 Operation: 'add', type of data: 'text'

09:29:20 12/02/2005 Attribute: 'objectClass'

09:29:20 12/02/2005 Value 0: 'top'

09:29:20 12/02/2005 Value 1: 'organizationalPerson'

09:29:20 12/02/2005 Value 2: 'person'

09:29:20 12/02/2005 Value 3: 'user'

09:29:20 12/02/2005 \*\*\*\*\* Modification data element: 1  
\*\*\*\*\*

09:29:20 12/02/2005 Operation: 'add', type of data: 'text'

09:29:20 12/02/2005 Attribute: 'sAMAccountName'

09:29:20 12/02/2005 Value 0: 'Methyl'

09:29:20 12/02/2005 \*\*\*\*\* Modification data element: 2  
\*\*\*\*\*

09:29:20 12/02/2005 Operation: 'add', type of data: 'text'

09:29:20 12/02/2005 Attribute: 'userPrincipalName'

09:29:20 12/02/2005 Value 0: 'Methyl@tools4ever.local3'

09:29:20 12/02/2005 \*\*\*\*\* Modification data element: 3  
\*\*\*\*\*

09:29:20 12/02/2005 Operation: 'add', type of data: 'text'

09:29:20 12/02/2005 Attribute: 'userAccountControl'

09:29:20 12/02/2005 Value 0: '512'

09:29:20 12/02/2005 \*\*\*\*\* Modification data element: 4  
\*\*\*\*\*

09:29:20 12/02/2005 Operation: 'add', type of data: 'binary'

09:29:20 12/02/2005 Attribute: 'unicodePwd'

09:29:20 12/02/2005 Binary data

09:29:20 12/02/2005 Adding LDAP directory service object  
'cn=Methyl,ou=Sales,dc=tools4ever,dc=local3' with LDAP modification data  
obtained from variable '%LdapData%'.

09:29:20 12/02/2005 LDAP directory service object  
'cn=Methyl,ou=Sales,dc=tools4ever,dc=local3' successfully added.

09:29:20 12/02/2005 \*\*\*\*\* Ready processing entry 105...

09:29:20 12/02/2005 Total number of script action execution errors: 0.

End of session

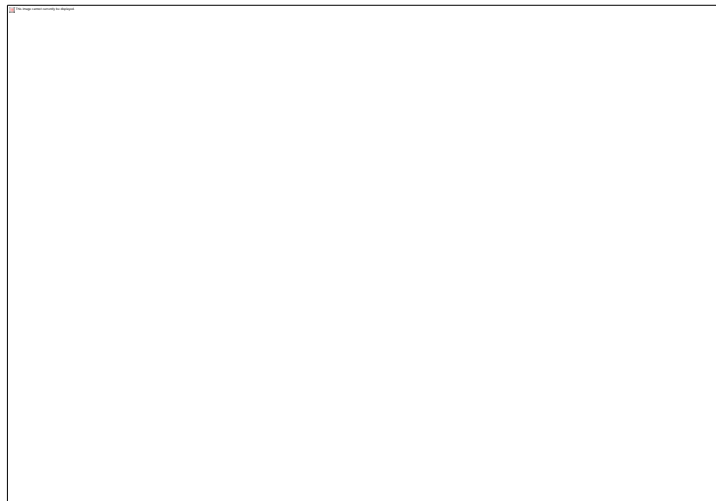
The log file first shows the following topics:

1. Initialization of the secure LDAP session;
2. Authentication of the connecting account;
3. Initialization of the LDAP modification data;
4. Creation of the user account.

### **Searching accounts and resetting passwords in Microsoft Active Directory using LDAP**

#### **Search and reset password**

This example project describes an UMRA application that searches the user accounts in an Active Directory domain. The accounts are shown in a form table. From the table, the end-user selects a user account and specifies a new password for the account. When the form is submitted, the password is reset.



*Figure 59: UMRA Forms client showing Reset password application.*

#### **Example project location**

The example project can be found at the following location, relative to the UMRA Console directory:

**.\\Example Projects\\LDAP\\ActiveDirectory\\SearchResetPassword**

The UMRA application contains the following projects:

Project	Description
LdapAd_Init	Initialize all of the variables used by the other projects of the application. This project is executed by the other projects.
LdapAd_Search	Search the user accounts in Active Directory using an LDAP query. The resulting user accounts are stored in a table that is passed to the next project of the wizard.
LdapAd_ResetPassword	Present the form to the end-user. When the form is submitted, reset the password of the selected user account.

*Table 12: Projects of the UMRA application to reset passwords*

The next sections describe each of the projects in detail.

#### **Project: LdapAd\_Init**

##### **Initialization project**

The project only contains a script, not a form. The project's script only sets a number of variables and it is executed by the other projects:

**LdapAd\_Search** and **LdapAd\_ResetPassword**.



*Figure 60: Variable initialization with project LdapAd\_Init.*

By using this method, the environment dependant variables need to be updated only in this project. The following variables are initialized by the project:

Variable	Example value	Description
%LdapServer%	king.tools4ever.local3	The DNS name of the Active Directory domain controller that runs the LDAP Server.
%LdapAccount%	cn=Administrator,cn=Users,dc=tools4ever,dc=local3	The full distinguished name of the administrative account that is authenticated on the domain controller. The account needs to have sufficient privileges to reset the password of the domain accounts.
%LdapPassword%	bA0U@HGWxUhz8MqG/+Uf23P#/qEDIG8A+	The password of the account that is authenticated. The password is stored encrypted in the UMRA script.

%SearchBase%	ou=Sales,dc=tools4ever,dc=local3	The part of the Active Directory tree from which user accounts must be obtained.
--------------	----------------------------------	--

Table 13: Variables initialized with project *LdapAd\_Init*.

When the script of the project is executed, the variables are initialized.

#### Project: **LdapAd\_Search**

##### Searching for user accounts

The project searches for the user accounts that must be presented to the end-user. The results are stored in a table variable that is shown in the form of project **LdapAd\_ResetPassword**. The project is configured as the initial project of project **LdapAd\_ResetPassword**.

The project only contains a script, not a form.

#### Script action: **Execute script**

##### Initializing variables by calling other script

In the first script action, the variable initialization script **LdapAd\_Init** is called to initialize the variables used in the subsequent script actions.



Figure 61: Script action: *Execute script* of project *LdapAd\_Init*.

The subsequent script actions do not use environment specified variable settings. To customize this UMRA application for another network environment, only the **LdapAd\_Init** script needs to be updated.

#### Script action: Setup LDAP session

##### Setup secure LDAP session

With this script action, the LDAP session is setup with the LDAP Server.



Figure 62: Script action: Setup LDAP session.

Since in subsequent script actions, a password of a user account is reset, the session must be setup using SSL encryption. This is a requirement from the Microsoft LDAP implementation. If SSL is not used, the search action will succeed, but the password reset action will always fail.

The LDAP session is setup using the values of the variables specified in project **LdapAd\_Init**. The resulting LDAP session is stored in variable **%LdapSession%** as specified by property **Ldap session**.

#### Script action: Search LDAP

##### LDAP search specification

In the next action, the search is performed. The search uses the initialized LDAP session (**%LdapSession%**) and returns the results in table variable **%LdapUsers%**.

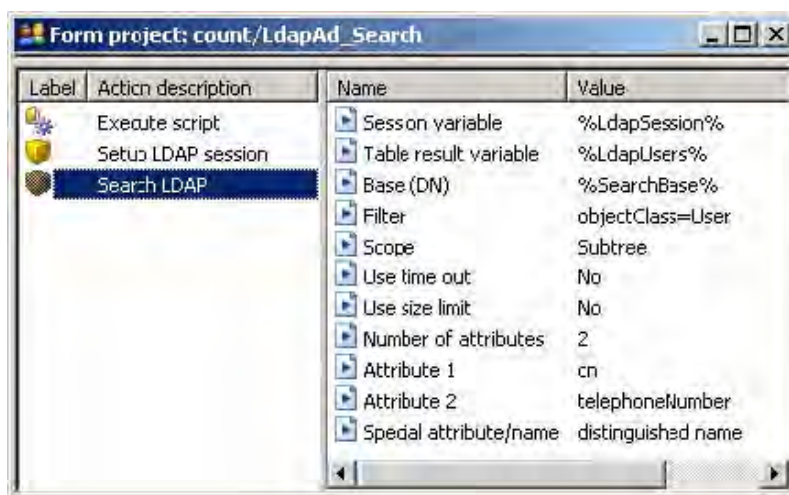


Figure 63: Script action: Search LDAP.

In this particular case, the search is performed in Active Directory subtree

**ou=Sales,dc=tools4ever,dc=local3**

as specified by variable **%SearchBase%**. All users found (**objectClass=User**) are returned. For each user account, the common name (**cn**), a phone number (**telephoneNumber**) and the **distinguished name** is obtained. The common name is needed to show to the end-user. The phone number is included to show how additional attribute values can be collected. The distinguished name is required to uniquely identify the user account, when the account is selected and the password of the account is reset.

When the action is executed, the table data is stored in output variable **%LdapUsers%**. This variable is used in the next project of the UMRA application.

**Project: LdapAd\_ResetPassword**

**Reset password project**



This is the main project of the UMRA application. It contains both a form and a script. The form shows the table with user accounts and the input fields for the new password. The script actually resets the password of the selected user account.

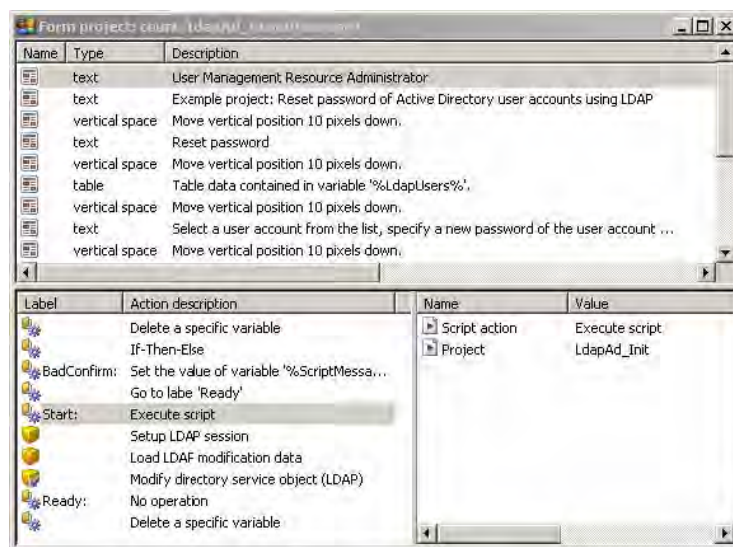


Figure 64: UMRA form project *LdapAd\_ResetPassword* with form and script to reset the password of an Active Directory user account using secure LDAP..

Besides a number of explanation text fields the form contains the following fields:

1. Table with user accounts
2. Two input fields for the new password

3. Button to submit the form.



**User Management Resource Administrator**  
Example project: Reset password of Active Directory user accounts using LDAP

### Reset password

Name	Phone
Actaia Byer	667-816-366
Actano Edeng	444-356-499
Actanth Dane	344-986-484
Actolym Actanth	312-947-415
Actoraë Aethyia	811-180-729

Select a user account from the list, specify a new password of the user account and click Reset password.

**New password:**

**Confirm password:**

Figure 65: Resulting form of ProjectLdapAd\_ResetPassword.

These fields are described in detail in the next sections.

**Form field: Table with user accounts**

**Variable generic table**

The table is defined as a **generic table** and with table type: **Variable**. To configure the table, the name of the variable and the names of the columns contains by the table variable must be specified.

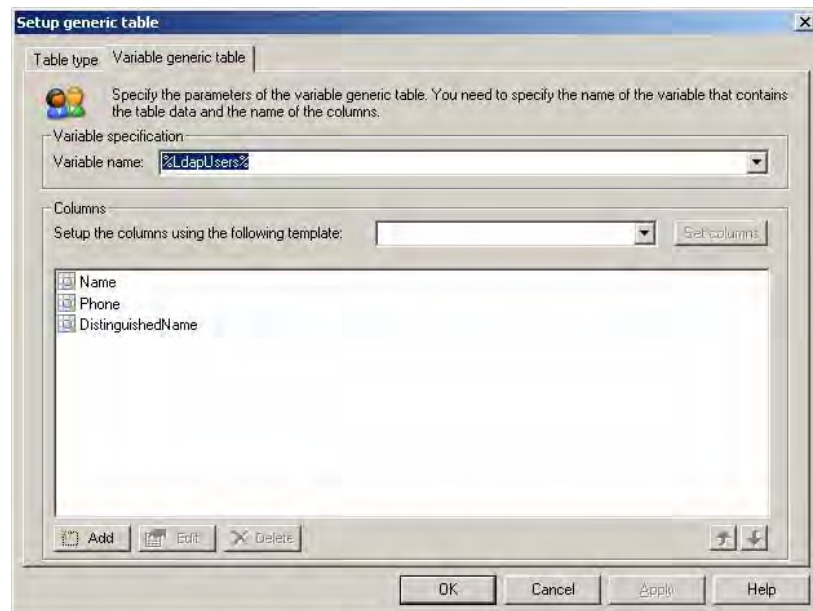


Figure 66: Generic table of the variable type. Specification of the name of the variable with table data and the name of the columns of the table.

The name of the variable corresponds with the name of the variable generated by script action **Search LDAP** of project **LdapAd\_Search**: **%LdapUsers%**. The project **LdapAd\_Search** is configured as the initial project of this project. The script of the initial project is executed just before the form of this project is created. In this application, the script of project **LdapAd\_Search** fills the variable **%LdapUsers%** with user accounts.

### Table columns and return variable

The variable `%LdapUsers%` only holds the table data, not the name of the columns. Therefore, the column names must be specified separately. When the generic table is configured, the table columns can be setup.

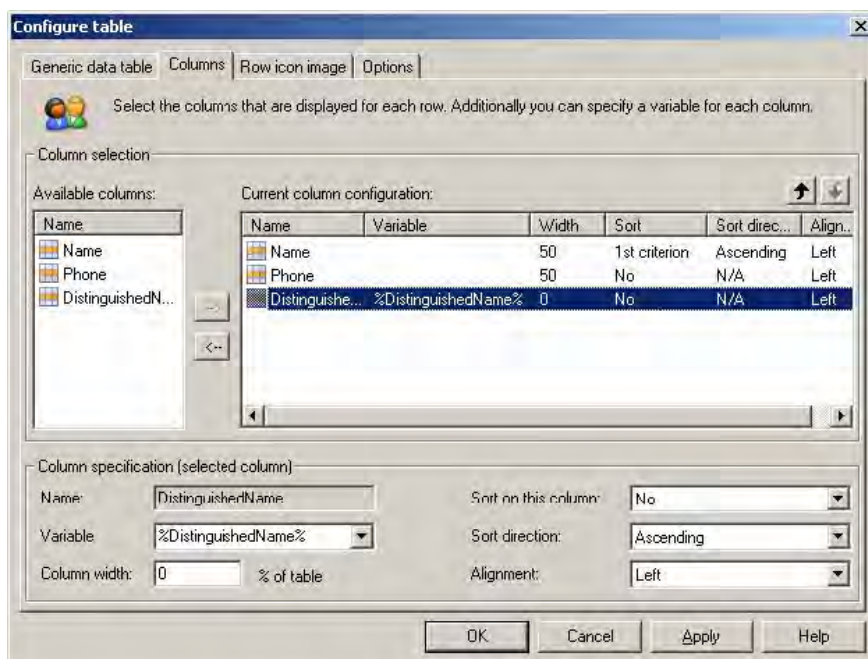


Figure 67: Specification of the columns shown in the form and the variable returned.

The **Columns** tab of the **Configure table** window shows the available columns and the configured columns. The available columns correspond with the columns specified for the generic table variable. In this example project, all three columns are configured. The column with the **DistinguishedName** has a zero width and an associated variable: `%DistinguishedName%`. The name does not look very user-friendly (example: `cn=John, ou=Sales, dc=tools4ever, dc=local2`). It therefore has a zero width, e.g. is not visible. When the end-user selects an account from the list, the distinguished name of the user account is copied into the variable and passed to the UMRA Service when the form is submitted.

### Form fields: Password text input fields

#### Password fields

Two input text fields are used to specify the new password. Initially, the input fields are empty. When the end-user has entered the password and confirmation field and the form is submitted, the values are copied to **%NewPassword%** and **%ConfirmPassword%** and sent to the UMRA Service.

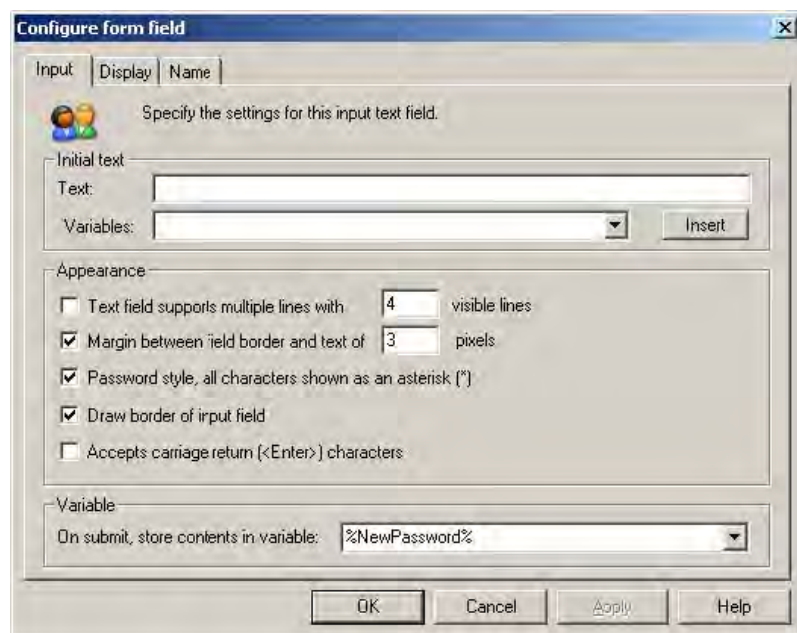


Figure 68: Password input text field specification.

The script of the project will process the specified values to reset the password.

#### **Form fields: Reset password submit button**

##### **Submit button actions**

When the user clicks the **Reset password** submit button, a number of actions are executed. First, the values of the form field variables are set according to the form input. This information is then sent to the UMRA Service.



Figure 69: Form actions executed by the UMRA Service when the Reset password button is clicked by the end-user.

The following actions are then executed by the UMRA Service:

1. **Check the input fields of the submitted form:** A check is performed to see if a user account is selected. If this is not the case, an error message is returned and the other script actions are not executed.
2. **Execute the script of the project that contains the form:** The script of the project is executed. The script resets the password of the selected user account and is described in the next sections.
3. **Return the form of project LdapAd\_ResetPassword:** The wizard starts over again and presents the screen with user accounts to the end-user.

The script of the project initializes a secure LDAP session with the LDAP Server and resets the password of the selected user account. The script is executed as one the button actions.

#### Script action: Delete a specific variable

When the form is submitted and processed, the same form is presented again. To reset the internal queue with variables, the **%ScriptMessage%** variable is reset. This variable is used to show a message to the end-user

and not automatically reset. Since the message from the previous session is no longer needed the variable must be reset. (If it was not reset, error messages from the previous session would still popup in the UMRA Forms client).



Figure 70: Script action: Delete a specific variable.

Note that all variables are reset as specified by the submit button action **Return the form of project LdapAd\_ResetPassword** with 2 exceptions: The variables %UmraFormSubmitAccount% and %ScriptMessage% are maintained.

#### Script action: If-Then-Else

##### Check password input

The script then checks the specified password and confirmed password with an **If-Then-Else** script action.

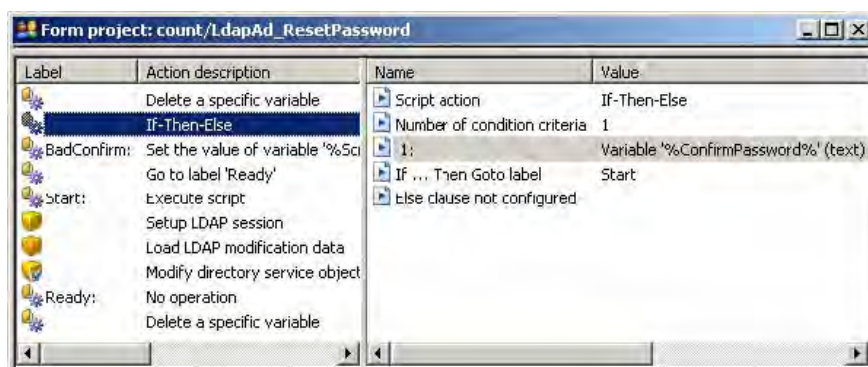




Figure 71: Script action: If-Then-Else to check the password and confirmed password

If the values of the variables **%ConfirmPassword%** and **%NewPassword%** correspond, execution of the script continues with the script action with label **Start**.

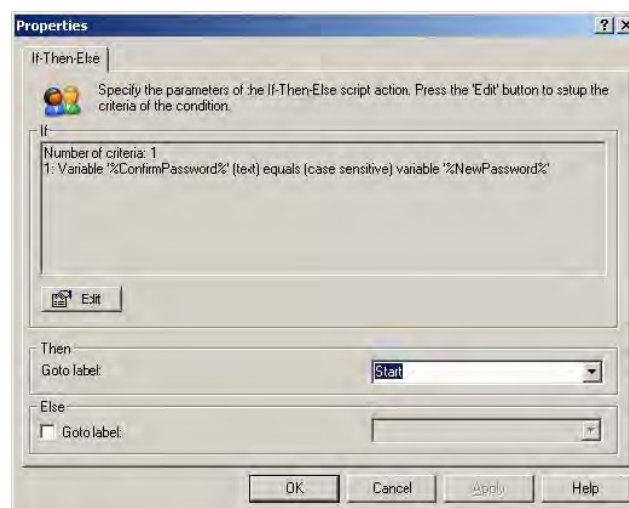


Figure 72: If-Then-Else specification.

If the values are not equal, an error message is generated by setting the variable **%ScriptMessage%** and jumping to the end of the script (Action: **Go to label Ready**).

When everything is fine, the script continues at the **Start** location.

**Script action: Execute script**

**Variable initialization**



The variable initialization script **LdapAd\_Init** is called to initialize the variables used in the subsequent script actions.



Figure 73: Script action to execute the initialization script.

When the variables are initialized, the LDAP session is setup.

#### Script action: Setup LDAP session

##### Setup secure LDAP session

With the action, the LDAP session is setup. The action properties are all specified by variables.

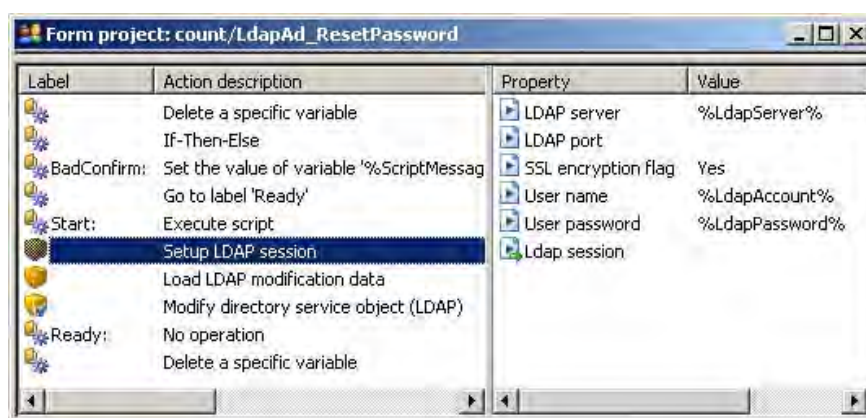


Figure 74: Script action to setup a secure LDAP session.

### SSL required to reset password

Note that the session must be setup with SSL. If SSL is not used, the LDAP Server will not accept the password reset action. This is a restriction enforced by the Active Directory LDAP Server implementation. When the session is initialized successfully, the session is stored in variable `%LdapSession%` as specified by the action property **Ldap session**.

### Script action: Load LDAP modification data

#### Password attribute specification

Next, the LDAP modification data used to reset the password is initialized. To reset a password, the attribute **unicodePwd** must be used



Figure 75: Script action to initialize the LDAP modification data with the password attribute value.

For the attribute, the type of modification is set to **Replace** since the existing password is replaced by a new one. The value of the attribute is set equal to the value of variable `%NewPassword%` as specified by the end-user. The resulting LDAP modification data is stored in variable `%LdapData%`.

Note: Internally, the **unicodePwd** attribute is handled a bit different compared to the other attributes, see knowledge base article KB269190 to check the details.

**Script action: Modify directory service object (LDAP)****Password reset action**

Finally, the password of the user account is reset.

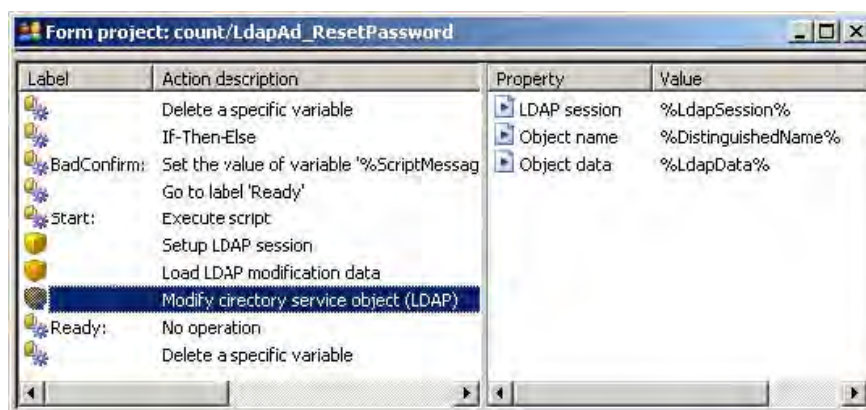


Figure 76: Script action to reset the password.

The action uses the following properties:

1. **LDAP session:** The LDAP session object, as initialized by script action **Setup LDAP session**.
2. **Object name:** The distinguished name of the directory service item of which one or more attribute values must be modified. In this case, the directory service item is the user account. The name of the account corresponds with the account selected in the form table with user accounts. The attribute value is specified with variable **%DistinguishedName%**. The variable is the output variable of the form table and determined when the user selects the **Reset** password submit button.
3. **Object data:** The modification data, specified by variable **%LdapData%** as determined by the previous action.

When executed successfully, the password is reset. Next a **No operation** action is executed. This action is used only as a script location reference. The action is jumped to when the entered password is incorrectly confirmed.

**Script action: Delete a specific variable****Variable cleanup**

When the script is executed, the wizard starts over again. To make sure that all variables are reset, the variable used for the selected user account is reset.



Figure 77: Script action to delete the %DistinguishedName% variable.

This action is not required since the variable is already reset by the action **Return the form of project LdapAd\_ResetPassword** specified for the **Reset password** submit button. When the script is used in another project, this might not be the case.

### Logging information

#### UMRA Service log

When executed successfully, the UMRA Service produces a log file as shown below.

Form is generated by the UMRA Service

```
09:11:22 12/09/2005 Form message: '12/09/2005,09:11:22,"SSP\J.
Vriens","Forms list",OK,N/A,"1 projects found for user 'SSP\J. Vriens'."
```

```
09:11:24 12/09/2005 Executing form initialization project 'LdapAd_Search'.
```

```
09:11:24 12/09/2005 Variable 1: %UmraFormSubmitAccount%=SSP\J. Vriens
```

```
09:11:24 12/09/2005 Calling project 'LdapAd_Init'. Executing script of project.
```

```
09:11:24 12/09/2005 Variable 1: %UmraFormSubmitAccount%=SSP\J. Vriens
```

UMRA Service connects to LDAP Server using SSL

09:11:24 12/09/2005 Setting up LDAP session with host 'king.tools4ever.local3'.  
Using SSL encryption: 'Yes'.

09:11:24 12/09/2005 User name:  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'.

09:11:24 12/09/2005 Secure LDAP session established with host  
'king.tools4ever.local3' (Protocol: 'TLS 1.0 client-side', encryption: 'RC4 stream',  
cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA', 2048 bits).

09:11:24 12/09/2005 Authenticating user  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'...

09:11:24 12/09/2005 User 'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'  
successfully authenticated on LDAP server host 'king.tools4ever.local3'.

09:11:24 12/09/2005 LDAP session information stored in variable  
'%LdapSession%'.

UMRA Service instructs LDAP Server to search for user accounts

09:11:24 12/09/2005 Searching LDAP with filter 'objectClass=User' in tree  
'ou=Sales,dc=tools4ever,dc=local3' using session variable '%LdapSession%'.  
Scope: 'Subtree'.

09:11:24 12/09/2005 No time out interval used.

09:11:24 12/09/2005 No size limit used.

219 user accounts found, form is presented to end-user

09:11:24 12/09/2005 Search action successfully completed, 219 entries found,  
stored in variable '%LdapUsers%'.

09:11:24 12/09/2005 Form message: '12/09/2005,09:11:24,"SSP\J.  
Vriens","Form load",OK,LdapAd\_ResetPassword,'

End-user selects a user account, enters password and clicks submit button

09:11:45 12/09/2005 Variable 1: %DistinguishedName%=CN=Actanth  
Dane,OU=Sales,DC=tools4ever,DC=local3

09:11:45 12/09/2005 Variable 2: %NewPassword%=hio78^

09:11:45 12/09/2005 Variable 3: %ConfirmPassword%=hio78^

09:11:45 12/09/2005 Variable 4: %UmraFormSubmitAccount%=SSP\J. Vriens

09:11:45 12/09/2005 Variable 5: %LdapServer%=king.tools4ever.local3

09:11:45 12/09/2005 Variable 6:

%LdapAccount%=cn=Administrator,cn=Users,dc=tools4ever,dc=local3

09:11:45 12/09/2005 Variable 7:

%LdapPassword%=bA0U@HGWxUhz8MqG/+Uf23P#/qEDIG8A+

09:11:45 12/09/2005 Variable 8:

%SearchBase%=ou=Sales,dc=tools4ever,dc=local3

09:11:45 12/09/2005 Variable 9: %LdapSession%=(0,0X0)

09:11:45 12/09/2005 Variable 10: %LdapUsers%=Table with 219 rows

09:11:45 12/09/2005 Variable 11: %NowDay%=09

09:11:45 12/09/2005 Variable 12: %NowMonth%=12

09:11:45 12/09/2005 Variable 13: %NowYear%=2005

09:11:45 12/09/2005 Variable 14: %NowHour%=09

09:11:45 12/09/2005 Variable 15: %NowMinute%=11

09:11:45 12/09/2005 Variable 16: %NowSecond%=45

09:11:45 12/09/2005 Deleting variable '%ScriptMessage%'.

UMRA Service compares entered password and confirmed password

09:11:45 12/09/2005 If-Then-Else condition [Variable '%ConfirmPassword%' (text) equals (case sensitive) variable '%NewPassword%'] result is TRUE, continue script execution with action 'Start'.

09:11:45 12/09/2005 Calling project 'LdapAd\_Init'. Executing script of project.

09:11:45 12/09/2005 Variable 1: %DistinguishedName%=CN=Actanth  
Dane,OU=Sales,DC=tools4ever,DC=local3

09:11:45 12/09/2005 Variable 2: %NewPassword%=hio78^

09:11:45 12/09/2005 Variable 3: %ConfirmPassword%=hio78^

09:11:45 12/09/2005 Variable 4: %UmraFormSubmitAccount%=SSP\J. Vriens

09:11:45 12/09/2005 Variable 5: %LdapServer%=king.tools4ever.local3

09:11:45 12/09/2005 Variable 6:

%LdapAccount%=cn=Administrator,cn=Users,dc=tools4ever,dc=local3

09:11:45 12/09/2005 Variable 7:

%LdapPassword%=bA0U@HGWxUhZ8MqG/+Uf23P#/qEDIG8A+

09:11:45 12/09/2005 Variable 8:

%SearchBase%=ou=Sales,dc=tools4ever,dc=local3

09:11:45 12/09/2005 Variable 9: %LdapSession%=(0,0X0)

09:11:45 12/09/2005 Variable 10: %LdapUsers%=Table with 219 rows

09:11:45 12/09/2005 Variable 11: %NowDay%=09

09:11:45 12/09/2005 Variable 12: %NowMonth%=12

09:11:45 12/09/2005 Variable 13: %NowYear%=2005

09:11:45 12/09/2005 Variable 14: %NowHour%=09

09:11:45 12/09/2005 Variable 15: %NowMinute%=11

09:11:45 12/09/2005 Variable 16: %NowSecond%=45

UMRA Service connects to LDAP Server using SSL

09:11:45 12/09/2005 Setting up LDAP session with host 'king.tools4ever.local3'.  
Using SSL encryption: 'Yes'.

09:11:45 12/09/2005 User name:

'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'.

09:11:46 12/09/2005 Secure LDAP session established with host

'king.tools4ever.local3' (Protocol: 'TLS 1.0 client-side', encryption: 'RC4 stream',  
cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA', 2048 bits).

09:11:46 12/09/2005 Authenticating user

'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'...

09:11:46 12/09/2005 User 'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'  
successfully authenticated on LDAP server host 'king.tools4ever.local3'.

09:11:46 12/09/2005 LDAP session information stored in variable

'%LdapSession%'.

LDAP modification data is setup with new password

09:11:46 12/09/2005 Storing LDAP modification data in variable '%LdapData%'.

09:11:46 12/09/2005 LDAP modification data:

09:11:46 12/09/2005 \*\*\*\*\* Modification data element: 0

\*\*\*\*\*

09:11:46 12/09/2005 Operation: 'replace', type of data: 'binary'

09:11:46 12/09/2005 Attribute: 'unicodePwd'

09:11:46 12/09/2005 Binary data

Password is reset

09:11:46 12/09/2005 Modifying LDAP directory service object 'CN=Actanth Dane,OU=Sales,DC=tools4ever,DC=local3' with LDAP modification data obtained from variable '%LdapData%'.

09:11:46 12/09/2005 LDAP directory service object 'CN=Actanth Dane,OU=Sales,DC=tools4ever,DC=local3' successfully modified.

09:11:46 12/09/2005 Deleting variable '%DistinguishedName%'.

Wizard starts over again

09:11:46 12/09/2005 Executing form initialization project 'LdapAd\_Search'.

09:11:46 12/09/2005 Variable 1: %UmraFormSubmitAccount%=SSP\J. Vriens

09:11:46 12/09/2005 Calling project 'LdapAd\_Init'. Executing script of project.

09:11:46 12/09/2005 Variable 1: %UmraFormSubmitAccount%=SSP\J. Vriens

09:11:46 12/09/2005 Setting up LDAP sessions with host 'king.tools4ever.local3'. Using SSL encryption: 'Yes'.

09:11:46 12/09/2005 User name:  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'.

09:11:46 12/09/2005 Secure LDAP session established with host 'king.tools4ever.local3' (Protocol: 'TLS 1.0 client-side', encryption: 'RC4 stream', cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA', 2048 bits).

09:11:46 12/09/2005 Authenticating user  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'...

09:11:46 12/09/2005 User 'cn=Administrator,cn=Users,dc=tools4ever,dc=local3' successfully authenticated on LDAP server host 'king.tools4ever.local3'.

09:11:46 12/09/2005 LDAP session information stored in variable '%LdapSession%'.



09:11:46 12/09/2005 Searching LDAP with filter 'objectClass=User' in tree 'ou=Sales,dc=tools4ever,dc=local3' using session variable '%LdapSession%'. Scope: 'Subtree'.

09:11:46 12/09/2005 No time out interval used.

09:11:46 12/09/2005 No size limit used.

09:11:46 12/09/2005 Search action successfully completed, 219 entries found, stored in variable '%LdapUsers%'.

09:11:46 12/09/2005 Next form project: 'LdapAd\_ResetPassword' ('LdapAd\_ResetPassword')

09:11:46 12/09/2005 Form message: '12/09/2005,09:11:45,"SSP\J. Vriens","Form submit",OK,LdapAd\_ResetPassword'

## **Updating group memberships in Microsoft Active Directory using LDAP**

### **Update group memberships**

This example describes a mass project to add user accounts to a specific Active Directory global group with UMRA using the LDAP actions instead of the normal UMRA Active Directory action.

The main purpose of the project is to show how to manage the different Active Directory object attributes to update user account group memberships.

Such a project can be used when a foreign domain is managed and the UMRA software runs on a computer that has no trust relationship with the domain. By using SSL, the application is completely secure.

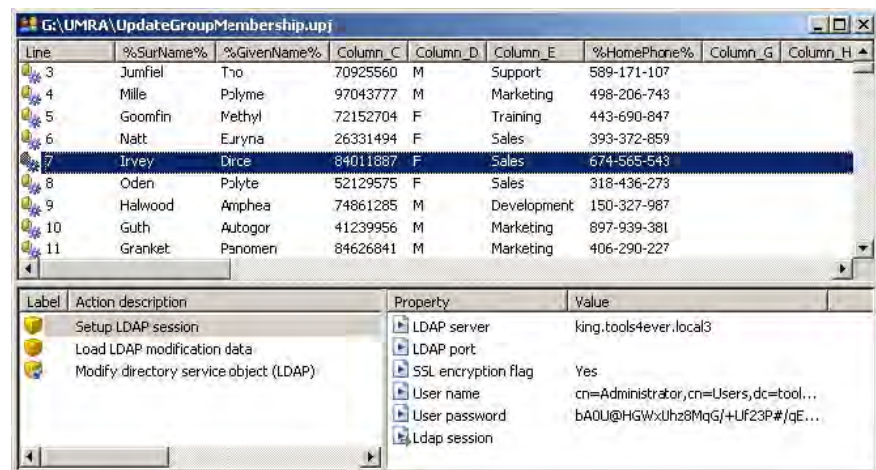
### **Example project location**

The example project can be found in the following location, relative to the UMRA Console program directory:

**.\\Example  
Projects\\LDAP\\ActiveDirectory\\GroupUpdate\\UpdateGroupMembershi  
p.upj**

The project contains embedded input data representing user accounts. The user accounts do exist in the Active Directory domain. For each line of the input data, the project repeats the following steps:

1. Setup a session with the LDAP Server running on the Active Directory domain controller.
2. Setup an LDAP modification data structure that contains all the attributes and values required to update the members of a specific group.
3. Update the group membership



The screenshot shows a project window titled "G:\UMRA\UpdateGroupMembership.upj". It contains a table with employee data and a list of actions for updating group membership.

Line	%SurName%	%GivenName%	Column_C	Column_D	Column_E	%HomePhone%	Column_G	Column_H
3	Jumfiel	Tho	70925560	M	Support	589-171-107		
4	Mille	Polyme	97043777	M	Marketing	498-206-743		
5	Goomfin	Methyl	72152704	F	Training	443-690-847		
6	Natt	Eurryna	26331494	F	Sales	393-372-859		
7	Irvey	Dirce	84011887	F	Sales	674-565-543		
8	Oden	Polyte	52129575	F	Sales	318-436-273		
9	Halwood	Amphea	74861285	M	Development	150-327-987		
10	Guth	Autogor	41239956	M	Marketing	897-939-381		
11	Granket	Panomen	84626841	M	Marketing	406-290-227		

Label	Action description	Property	Value
	Setup LDAP session	LDAP server	king.tools4ever.local3
	Load LDAP modification data	LDAP port	
	Modify directory service object (LDAP)	SSL encryption flag	Yes
		User name	cn=Administrator,cn=Users,dc=tool...
		User password	bA0U@HGwXUhz8MqG/+Uf23P#/qE...
		Ldap session	

Figure 78: UMRA mass project to update group memberships

The next sections describe the project in detail.

#### Script action: Setup LDAP session

The session is setup with the LDAP Server running on the domain controller. In this example project, the LDAP server is

**king.tools4ever.local3**

When connected, a user account is authenticated. The user account must have sufficient access rights in the Active Directory domain to update the group memberships. Although not required for this type of operation, it is recommended to initialize a secure session with SSL. This is particularly important since the administrative account password is send over the line.

When successfully established, the session is stored in variable **%LdapSession%** as specified by property **Ldap session**.

**Script action: Load LDAP modification data**

Update *member* attribute of *group*

When the user account is added to a group, two Active Directory items are updated:

1. The user account is now a member of the group. In LDAP: the **memberOf** attribute of the **user account** contains an extra value: the distinguished name of the group.
2. The group now has an extra member. In LDAP: the **member** attribute of the **group** contains an extra value: the distinguished name of the user account.

In Active Directory, the update must be performed by changing the **member** attribute of the **group**. The **memberOf** attribute of the user account is a so called computed back-link attribute and cannot be used to update user account group memberships.

When the membership is updated, the **memberOf** attribute of the **user account** is updated automatically.

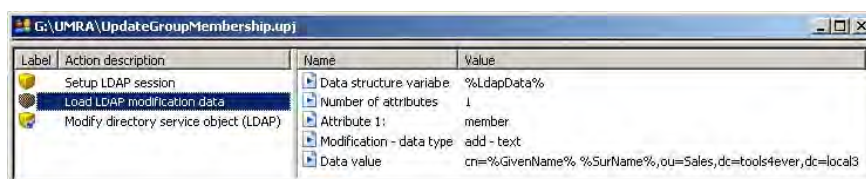


Figure 79: Script action to setup LDAP modification data for the group

So the LDAP modification data must specify the new value for the **member** attribute of the group: the distinguished name of the user accounts. In the example project, the distinguished name of the user account is

**cn=%GivenName% %SurName%,ou=Sales,dc=tools4ever,dc=local3**

The variables **%GivenName%** and **%SurName%** are taken from the input file.

The LDAP modification data is stored in variable **%LdapData%** that is used by the next action.

**Script action: Modify directory service object (LDAP)**

Finally, the user account is added to the group by updating the **member** attribute of the **group**.

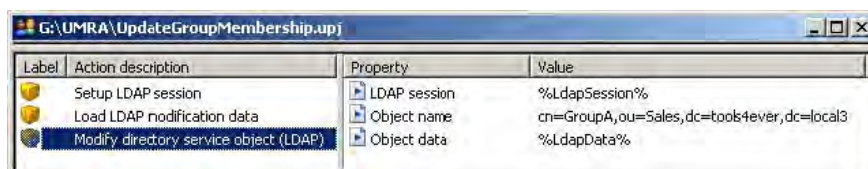


Figure 80: Script action to add the user account to the group by updating a group attribute.

In the example project, the group is specified as:

**cn=GroupA,ou=Sales,dc=tools4ever,dc=local3**

**Add user account to group**

The example project shows how to add a user account to a group. To remove a user account from a group, a similar script must be used but the modification data must be specified as a **Delete modification type**.

**Logging information****UMRA Console log**

When executed successfully, the UMRA Console produces a log file with the following contents.

Starting User Management Resource Administrator session, build 1213 at  
10:58:09 12/09/2005

10:58:09 12/09/2005 \*\*\*\*\* Processing entry 22...

10:58:09 12/09/2005 Variable 1: %GivenName%=Neme

10:58:09 12/09/2005 Variable 2: %SurName%=Cafer

10:58:09 12/09/2005 Variable 3: %HomePhone%=496-411-709

10:58:09 12/09/2005 Variable 4: %NowDay%=09

10:58:09 12/09/2005 Variable 5: %NowMonth%=12

10:58:09 12/09/2005 Variable 6: %NowYear%=2005

10:58:09 12/09/2005 Variable 7: %NowHour%=10

10:58:09 12/09/2005 Variable 8: %NowMinute%=58

10:58:09 12/09/2005 Variable 9: %NowSecond%=09

Setup secure LDAP session

10:58:09 12/09/2005 Setting up LDAP sessions with host  
'king.tools4ever.local3'. Using SSL encryption: 'Yes'.

10:58:09 12/09/2005 User name:  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'.

10:58:09 12/09/2005 Secure LDAP session established with host  
'king.tools4ever.local3' (Protocol: 'TLS 1.0 client-side', encryption: 'RC4 stream',  
cipher strength: 128 bits, hash: 'MD5', 128 bits, key exchange: 'RSA', 2048 bits).

10:58:09 12/09/2005 Authenticating user  
'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'...

10:58:09 12/09/2005 User 'cn=Administrator,cn=Users,dc=tools4ever,dc=local3'  
successfully authenticated on LDAP server host 'king.tools4ever.local3'.

10:58:09 12/09/2005 LDAP session information stored in variable  
'%LdapSession%'.

Setup LDAP modification data to update member attribute of group

10:58:09 12/09/2005 Storing LDAP modification data in variable '%LdapData%'.

10:58:09 12/09/2005 LDAP modification data:

10:58:09 12/09/2005 \*\*\*\*\* Modification data element: 0  
\*\*\*\*\*

10:58:09 12/09/2005 Operation: 'add', type of data: 'text'

10:58:09 12/09/2005 Attribute: 'member'

10:58:09 12/09/2005 Value 0: 'cn=Neme  
Cafer,ou=Sales,dc=tools4ever,dc=local3'

Add user account to group

10:58:09 12/09/2005 Modifying LDAP directory service object  
'cn=GroupA,ou=Sales,dc=tools4ever,dc=local3' with LDAP modification data  
obtained from variable '%LdapData%'.

10:58:09 12/09/2005 LDAP directory service object  
'cn=GroupA,ou=Sales,dc=tools4ever,dc=local3' successfully modified.

10:58:09 12/09/2005 \*\*\*\*\* Ready processing entry 22...

10:58:09 12/09/2005 Total number of script action execution errors: 0.

End of session

### **3.6.8. References**

*LDAP System Administration*, by Gerald Carter, O'Reilly Media Inc. 2003

*Debian GNU/Linux 3.1 Bible*, by Benjamin Mako Hill, Wiley Publishing, Inc.  
2005

*Novell's Guide to Netware 6 Networks*, by Jeffrey F. Hughes and Blair W.  
Thomas, Wiley Publishing, Inc. 2002

---

### 3.7. Name generation

In Active Directory, a user has many different names. Some of these names have to be unique, which means that you cannot have another user account with the same name. UMRA can generate all user names used in Active Directory, including these unique user names. To generate these names automatically, UMRA uses name generation algorithms.

Before you start reading this guide, you should have a basic understanding of the concept of variables. For more information, see *UMRA Basics* on page 3.



*Read the full PDF version of Name Generation*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/umra-name-generation.pdf>

### **3.7.1. Generating user names**

In UMRA, you can either use the built-in name generation algorithms to generate user names or create your own algorithms. Both methods are explained in the following sections.

#### **Using the built-in name generation algorithms**

In Active Directory, many different user names can be found:

- CommonName
- sAMAccountName – Must be unique within the domain
- User-Principal-Name - this internet style login name is the SAM Account Name combined with the domain it is defined within, making it unique within a forest.
- Given-name
- DisplayName
- Etc.

UMRA comes with a set of built-in name generation algorithms to generate all these names automatically. A name generation algorithm is a set of rules which defines the following:

- How one or more names can be composed from other names. By default, the first, middle and last name of a user are used to generate the above mentioned user names.;
- How the resulting names can be made unique.

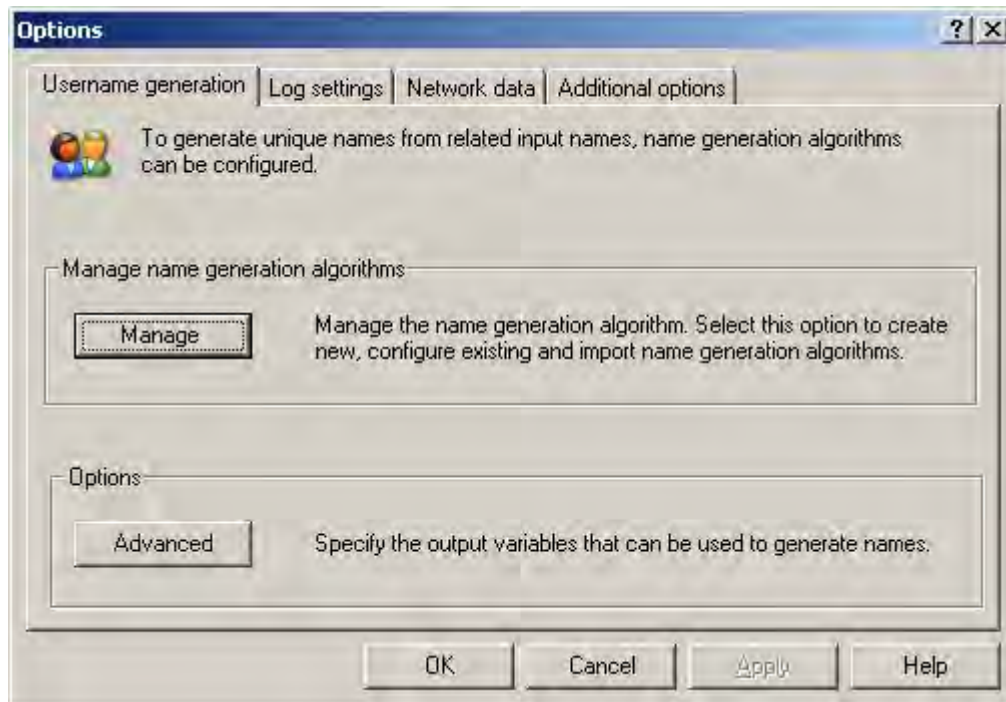
This set of rules can be defined in many different ways and completely in compliance with company naming conventions.

#### **Generating Active Directory names based on the user's first, middle and last name**

By default, a name generation algorithm can generate all user names in Active Directory based on the first, middle, and last name of a user. To see the various name generation results when using the standard name generation algorithms in UMRA, you can perform the following test:

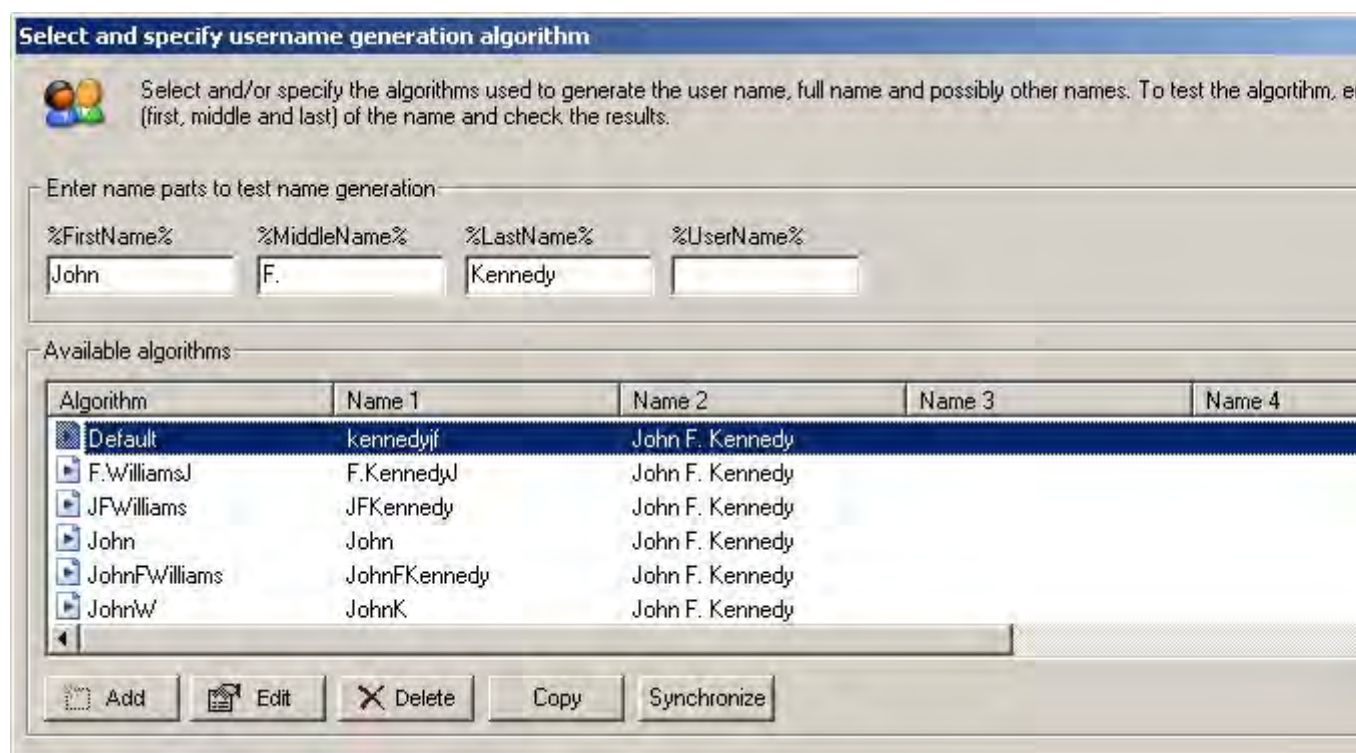


1. In the **UMRA Console**, select **Tools→Options** and click the **Manage** button in the **Manage name generation algorithms** section.



The **Select and specify user name generation algorithm** comes up, in which example values are shown for the input names, a list with available algorithms and the results of these algorithms according to the specified values of the input names.

- Enter the names “John”, “F.”Kennedy” in the %FirstName%, %MiddleName% and %LastName% fields respectively to see the results of each of the available name generation algorithms.



Note that by default, UMRA generates two output names, which are displayed in the Name1 and Name2 columns respectively. Entering “John”, “F.”Kennedy” results and choosing the **Default** name generation algorithm results in the following two output names:

- Name1 = kennedyjf
- Name2 = John F. Kennedy

Together with the user’s first, middle and last name, these output names can be used to specify all user names for an account.

### Specifying input names

The first, middle and last name of a user can be obtained in various different ways:

1. Read directly from an input file



2. User input in a form



3. Database query

#### Using variables to specify input names

In practice, UMRA uses variables to specify input (and output) names. The input from a CSV file, form or database is linked to a variable (e.g. %FirstName%, %MiddleName% and %LastName%). In turn, these variables can be used for the following purposes:

- to provide input for the name generation algorithm to generate other user names;
- to specify some of the user names in Active Directory.

If the first, middle and last name are entered by the end user in a form for instance, this information can be linked to the variables **%FirstName%**, **%MiddleName%** and **%LastName%**. With the value of these variables you can specify the following user names:

<b>Name attribute Active Directory</b>	<b>Variable</b>
Given-name	<b>%FirstName%</b>
Initials	<b>%MiddleName%</b>
Surname	<b>%LastName%</b>

### Using variables to specify output names

Based on the value of the variables **%FirstName%**, **%MiddleName%** and **%LastName%**, the default name generation algorithm will generate the remaining user names and store the result in the variables **%FullName%** and **%Username%**. These variable values can then be used to set various name attributes as shown in the table below:

<b>Name attribute Active Directory</b>	<b>Variable</b>
CommonName	<b>%FullName%</b>
sAMAccountName	<b>%UserName%</b>
User-Principal-Name	<b>%UserName%@&lt;Domain_name&gt;</b>
DisplayName	<b>%FullName%</b>

### Customizing name generation algorithms

In some cases it is not sufficient to use the built-in name generation algorithm. This could be the case when your company has specific requirements regarding naming conventions. For such cases, it is possible to customize the way in which an output name is generated.

### Name generation methods

A name generation method performs the following tasks:

1. it specifies in detail how a single output name is generated based on one or more input names (e.g. first name, middle name, last name).

2. it can ensure the uniqueness of a user name. This is explained in more detail in Using a method to create unique user names.

#### Using a method to generate output names

To generate an output name (e.g. %UserName%, %FullName%), a name generation algorithm uses a name generation method.

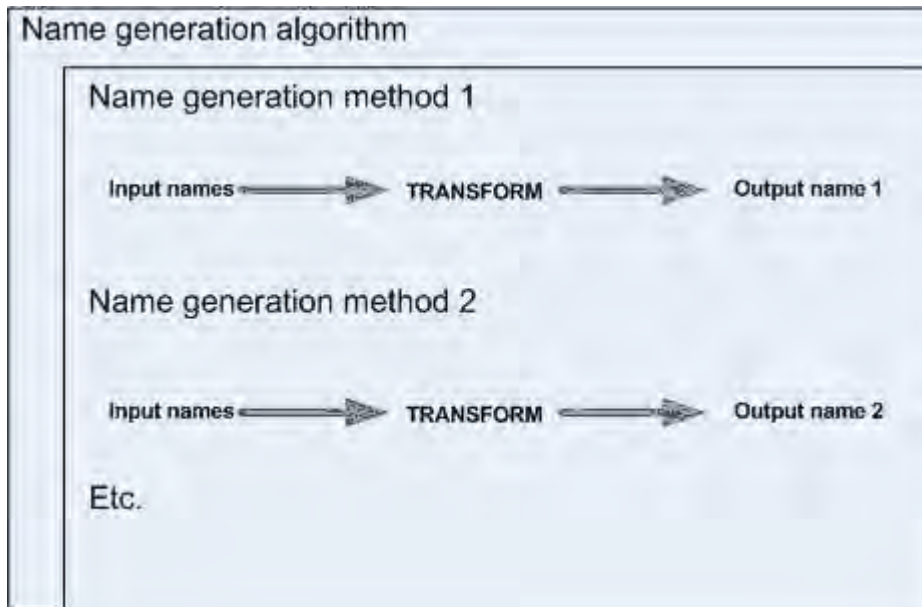


Figure 1 – A name generation method uses input names to construct a new output name

A method could take the following input names, for instance:

%FirstName% = John

%MiddleName% = F.

%LastName% = Kennedy

and transform these names into a new output name which can be created to specify, amongst others, the SAMAccountName:

jfkennedy

You will clearly see that in this case:

“John” must be transformed to “j”

“F.” must be transformed to “f”

“Kennedy” must be transformed to “kennedy”.

These transformations are achieved by using formatting functions. UMRA comes with a wealth of formatting functions to change the value of a variable.

The table below shows which formatting functions can be used for the method described above to obtain the new name parts.

Input name	Formatting function(s)	Example	Result
%Firstname%	Shorten name: Convert to the first character of the name Case conversion: convert to lowercase	John  J	J  j
%Middlename%	Shorten name: Convert to the first character of the name Case conversion: convert to lowercase	F.  F	F  f
%Lastname%	Case conversion: convert to lowercase	Kennedy	kennedy

The new name parts “j”, “f” and “kennedy” together make up the new output name.

#### Using a method to create unique user names

In many cases you will want to create output names which are unique. When creating user accounts in Active Directory for instance, there are two user name attributes which must be unique:

- SAM Account Name
- User Principal Name

To ensure the uniqueness of user names, you can specify an iteration for a name generation method. The most simple form of iteration is the addition of a sequential number at the end of a generated user name (e.g. johnw1, johnw2, johnw3, etc.). Using iteration, the same method can be applied again and again until the generated output name has been made unique (see figure 2).

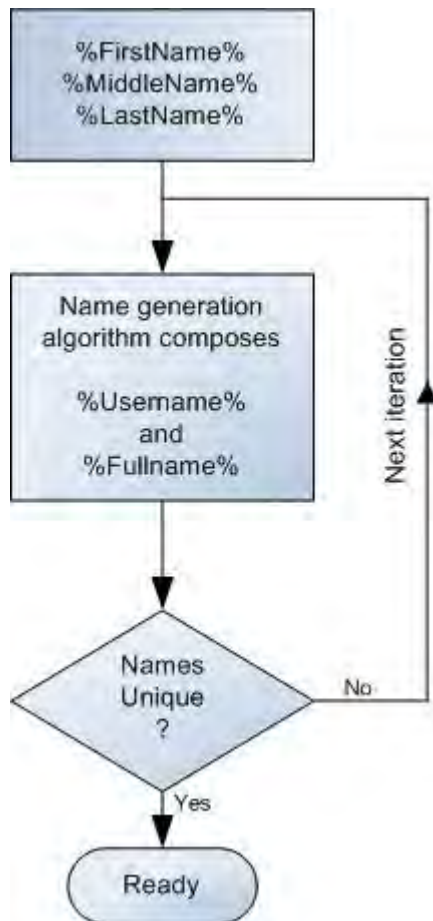


Figure 2 – Generating unique output names

The number of iterations for each method and the iteration sequence can be specified.

If the generated output name is not unique after one pass, a new output name will be generated using the same method with the specified iteration sequence. If the number of iterations has exhausted the specified number of iterations for the method, the algorithm will continue with the next method. This process will stop when a unique name has been generated.

---

### 3.8. UMRA tables

In UMRA, the use of tables is an important instrument for managing user accounts and associated resources, both in Active Directory and other information systems. This concept is explained in detail in this section.



*Read the full PDF version of UMRA tables*

<http://www.tools4ever.com/resources/pdf/user-management-resource-administrator/Umra-Tables-User-Guide.pdf>



*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### 3.8.1. Introduction

Many employees nowadays have access to a wide variety of systems and applications. Some examples are access to a company's intranet, phone systems, HR systems, printers, etc. User accounts and associated resources are usually maintained in Active Directory (or other directory services). Larger organizations often have multiple information systems, in which case user resource data are also stored outside Active Directory (a SQL server holding HR data, phone book applications, location systems, etc.). Using UMRA, an administrator can create projects to deal with virtually any user management task.

In UMRA, the use of tables is an important instrument for supporting this concept of managing users and associated resources in Active Directory and other information systems, by facilitating the following tasks :

**Managing and selecting user accounts, resources and other input fields** - In case of a delegation project, a list table can be included in a form window to display and select fixed data (a list table), Active Directory data (LDAP query), results of an NT 4 network call or database data. Some examples:

to select a user for whom the password needs to be reset. Figure 1 shows a form table listing the users obtained through an LDAP query on Active Directory.



The screenshot shows a 'Form preview' window with a title bar containing a bell icon and the text 'Form preview'. The main content area has a header with two user icons and the title 'Reset Password'. Below the header is a table with two columns: 'Common name' and 'Username'. The table contains three rows of user data. Below the table are two password input fields labeled 'Password:' and 'Confirm password:', each with a masked password 'xxxxxxxx'. At the bottom left is a button labeled 'Reset Password'.

Common name	Username
Henry J. Kissinger	hjkissinger
Philip Worth	pworth
Rodney Smith	rsmith

Figure 1 - Delegated project for resetting a password

to show a list of services or printers you wish to manage. A script action can then be executed for the selected table entry (see Figure 2) ;

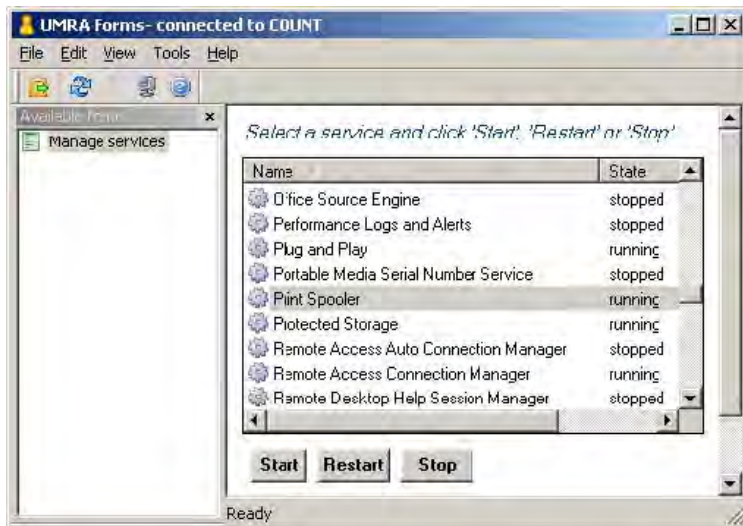


Figure 2 - List of services as part of a delegation project to manage services (e.g. stop, start, pause, resume services)

to select data from a database (e.g. a list of departments or a table containing corporate telephone numbers)



Figure 3 - List of departments as a result of a database query

**Manage the processing of tabular data (row by row)** – this method is used to run actions against each row of a table. It can be used for instance, to query Active Directory for all the groups of which a user is a member and to perform an action for each group in the resulting table (e.g. setting a new group membership and removing the existing one(s)).

**Bulk data processing** - Tables are also used to facilitate the mass update of user data (e.g. mass creating Exchange mailboxes or bulk create and edit users as part of a migration project). In such mass projects, where a table is based on an imported CSV file, script actions can be executed for each row in a table (containing user resource objects, for instance). This specific use of a table will not be discussed in this document.

The last chapter of this user guide contains several hands-on examples to get familiar with the various table types. Each hands-on will take approximately 30 minutes to complete.

### 3.8.2. The concept of tables in UMRA

#### General

With UMRA, solutions can be created for the delegation of one or more user management tasks. Forms & Delegation includes two separate applications, the **UMRA Console** and **UMRA Forms**. In this document, it is assumed that you already have basic knowledge of working with UMRA. If this is not the case, please make yourself familiar first with the UMRA basics.

The **UMRA Console** is used by systems administrator to build an interface for the end user and to add and configure built-in script actions for the delegation project. The interface is created using forms which can be fully customized with your own titles, text fields, user input fields, buttons, tables, graphics, etc. Once the form project has been properly set up, the delegate user can run it in the UMRA Forms application to perform a specific user management task. The underlying scripting intelligence is neither accessible nor visible for the delegate user.

#### Selecting data using form tables

In a form, the administrator can also add form tables. Form tables are used to display user resource data from Active Directory and other information systems.



CommonName	UserName
Henry Bush	hbush
John Henderson	hendj
Kathy Johnson	kjohnson
John Smith	smithj
Geoff Marlowe	gmarlowe

Password:

Confirm password:

**Reset Password**

Figure 4 - LDAP table showing users in Active Directory as part of a Reset Password delegation form

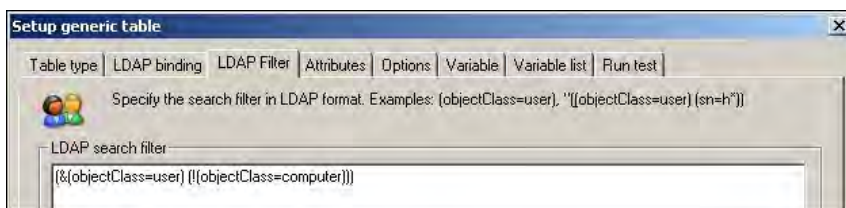
In the UMRA Console, the administrator specifies which script actions should be run on the selected table entries. The execution of these script actions can be assigned to a button (e.g. "Reset Password").

In UMRA, a wide variety of script actions is included to create, manage and delete Active Directory objects. Finally, the administrator also specifies who has privileges to run the project.

In figures 5 and 6, the use of a form table is shown in more detail. As part of a delegation project to reset passwords, an LDAP table is inserted in the form window to retrieve all users in a specific organizational unit in Active Directory using the query

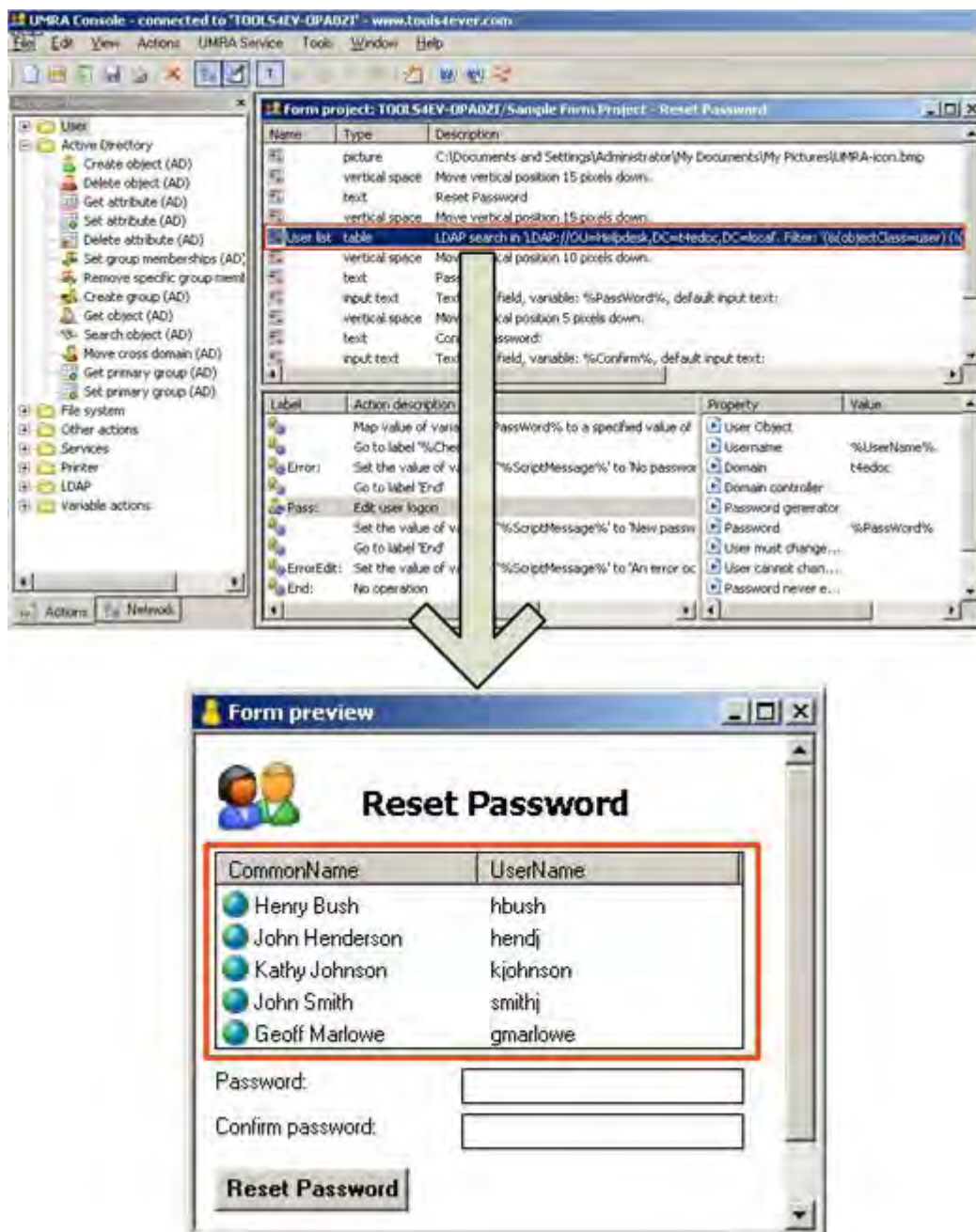
**(&(objectClass=user)(!(objectClass=computer)))**

with a binding to LDAP://OU= <NameOU>, DC=MyDomain, DC=local.



*Figure 5 - Defining an LDAP query for an LDAP table*

This query retrieves all objects of the “user” object class in the OU “<NameOU>”. Computer accounts are exempted from the results. In the resulting form table (Figure 6), a user can be selected for whom the password needs to be reset.



*Figure 6 - Reset password delegation project. The result of the LDAP query is displayed in the LDAP table (shown here with a red marquee)*

### Form table types

In a project form, several table types are available which are shown in the table below:

Table type	Subtype	Description
Network table		Used to obtain the user accounts of an OU, global group, domain or single computer using an NT network call.
Fixed table		Used to display a list of fixed content in a table (e.g. a list of department names).
Generic table	LDAP	Used to show the results of an LDAP query in a table.
Generic table	Database	Used to show the results of a database query in a table.
Generic table	Variable	See section <i>Special table type - Generic table Variable</i> on page 16 for more information.

#### *Network table*

A network table is used for Windows NT 4 environments and can be of one of the following network data types:

- User accounts of an OU,
- User accounts of a global group,
- User accounts of a domain
- User accounts of a single computer.

Specifying the network data type **only** determines the scope of the network calls to be executed by UMRA and the columns that can be shown for the corresponding network data table. To determine the data that must be collected, you must specify the actual parameters or arguments that are used to collect the network data. These are different for each network data type (see the table below).

Network table	Scope	Arguments	Columns
OU	User accounts of one or more OUs (including child OUs)	<Domain>/<OU>/<OU> or LDAP name of the object.	Common name Description Username Display name Domain OrganizationalUnit Object distinguished name

Network table	Scope	Arguments	Columns
Global group	User accounts that are a member of one or more groups	<Domain>\<GlobalGroup> or <DomainController>\<GlobalGroup>	Full name Description Username Global group Domain
Domain	User accounts in a domain	Domain name (NETBIOS or DNS format)	Full name Description Username Domain
Single computer	User accounts that are maintained on a computer (not necessarily a domain controller).	Computer name (NETBIOS or DNS format)	Full name Description Username Computer Domain Type

The columns can be assigned to a variable. When a delegate user selects an entry in the resulting network table, the name of the variable and the corresponding value will be passed to the UMRA service. See the section *Processing user input* on page 20 for more information.

#### *Generic table - LDAP query*

This powerful generic table type allows you to query Active Directory and show the results in a form table. To run a (complex) LDAP search, the following information should be specified:

1. **LDAP binding** - the scope of the LDAP search;
2. **LDAP filter** - the objects you wish to filter on;
3. **LDAP Attributes** – the attributes you wish to retrieve for these objects. These components will be individually discussed.

#### LDAP binding

Starting with Windows 2000, the LDAP provider is used to access Active Directory. This binding method requires a binding string, which can be defined in three different ways in UMRA:



Binding method	Description
Global Catalog	The global catalog is a searchable master index containing directory data of all domains in a forest. It contains an entry for every object in the forest, but it does not include all properties of each object. The Global Catalog is used to improve the response time of LDAP searches. The properties included in the Global Catalog are generally useful for searches and are considered static.
Active Directory root	This option will bind to the Active Directory root of a domain controller. The Active Directory contains all the network information for the forest. This binding method is suitable for retrieving dynamic properties.
Manual	You can also enter a binding string yourself. This binding string is the AdsPath of an object in Active Directory, consisting of the LDAP provider moniker (LDAP://) appended to the Distinguished Name of the object. The Distinguished Name specifies both the name and the location of an object in the Active Directory hierarchy.

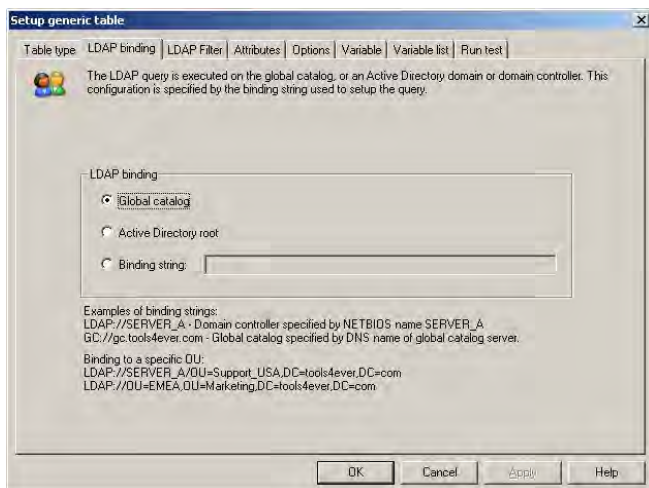


Figure 8 - LDAP binding options to bind to Active Directory

#### LDAP filter

An LDAP search filter can be defined as a clause specifying the conditions that must be met by Active Directory objects. Only those objects meeting the requirements are returned.

A condition takes the form of a conditional statement, such as "(cn=TestUser)". Each condition must be enclosed in parenthesis. In general, a condition includes an attribute and a value, separated by an operator.

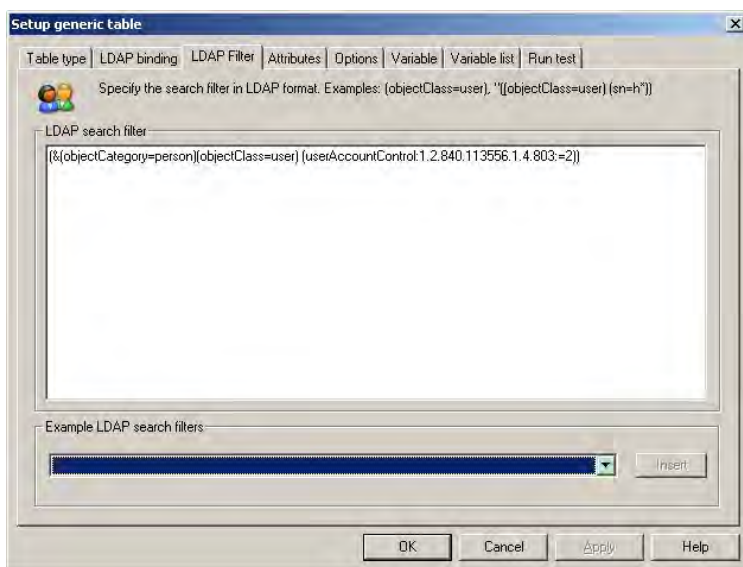
Conditions can be combined using the following operators (note that the operators "<" and ">" are not supported).

Operator	Description
=	Equal to
~=	Approximately equal to
<=	Less than or equal to
>=	Greater than or equal to
&	AND
	OR
!	NOT

Conditions can also be nested using parenthesis. Furthermore, you can use the "\*" wildcard character in the search filter.

An example of an LDAP filter is shown in figure 9:

**(&(objectCategory=person)(objectClass=user) (userAccountControl:1.2.840.113556.1.4.803:=2))**



*Figure 9 - Defining an LDAP filter*

This filter, used to obtain disabled user objects, includes 3 conditions, which should all be met (indicated by the AND ("&") operator).

The **userAccountControl** attribute in this example needs some further explanation. This is a so called bitmask attribute, a single attribute containing numerous property values for controlling user account behaviour. A bitmask attribute can be evaluated using an LDAP matching rule using the following syntax:

*attributename:ruleOID:=value*

where *attributename* is the LDAPDisplayName of the attribute, *ruleOID* is the object ID (OID) for the matching rule control, and *value* is the decimal value you want to use for comparison. In the example above, the string “1.2.840.113556.1.4.803=2” represents an LDAP matching rule for disabled user accounts.

#### LDAP Attributes

Each object in Active Directory has a set of attributes, defined by and depending on its type and class. Using the LDAP filter you have filtered on some objects representing single entities (users, computers, printers, applications, etc.) and their attributes. In the **Attributes** tab you can define the LDAP display name for the attributes you wish to return for the filtered objects. In the example shown in figure 10, the attributes **Name**, **Description** and **sAMAccountName** are specified as the attributes to be returned.

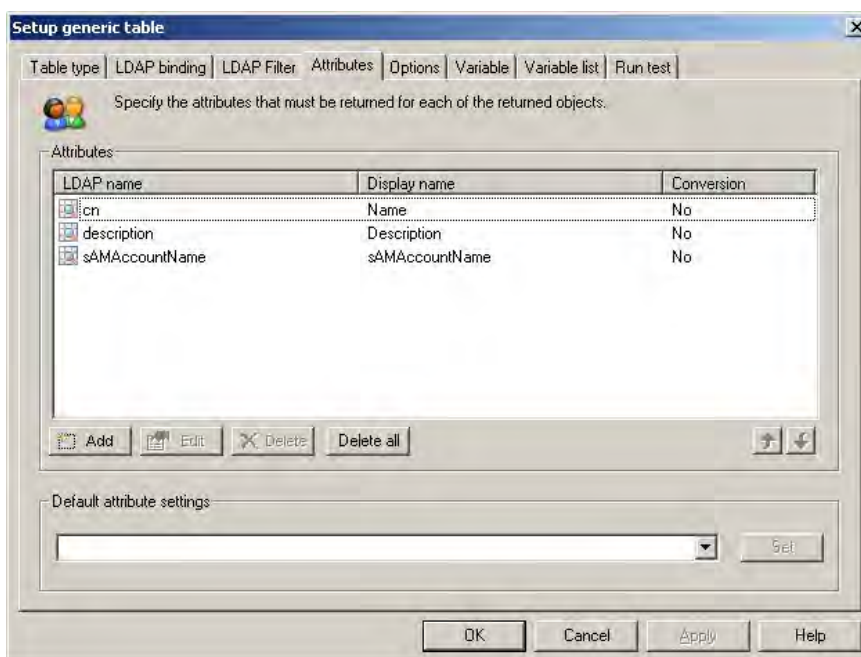


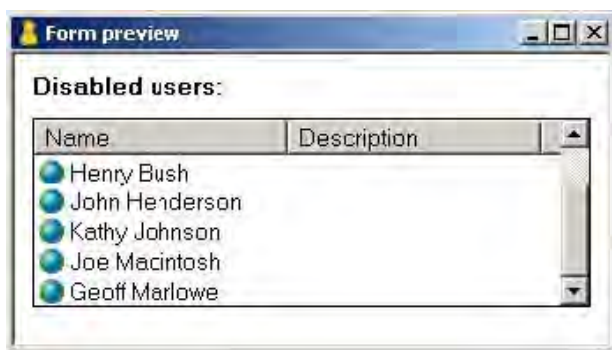
Figure 10 - Specifying attributes to be returned for filtered objects

Attributes can either be selected from an extensive list of built-in attributes, or entered by the user. For a full list of attributes, see <http://msdn.microsoft.com/library/default.asp?url=/library/en->

[us/adschema/adschema/attributes\\_all.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes_all.asp) [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes\\_all.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes_all.asp) or check the MSDN library.

The final result of an LDAP search is displayed in a generic table (see figure 11). In the same way, you can retrieve user accounts in a specific OU, show a list of groups, etc.

To process selected table entries, the column holding the attribute values can be assigned to a variable and used as input for a script action. For more information, see section *Processing user input* on page 20.



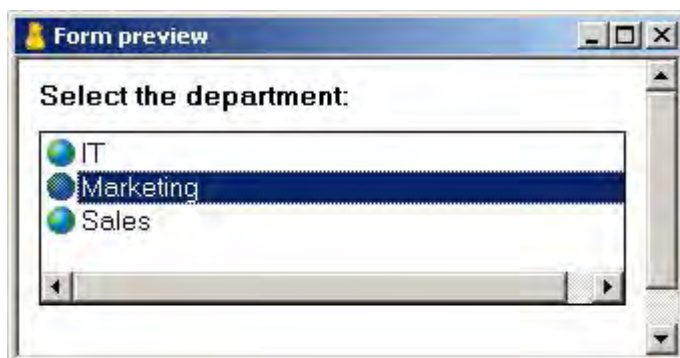
The image shows a 'Form preview' window with a title bar containing a yellow icon, the text 'Form preview', and standard window controls. The main content area is titled 'Disabled users:' and contains a table with two columns: 'Name' and 'Description'. The table has five rows of data, each preceded by a blue globe icon. The names listed are Henry Bush, John Henderson, Kathy Johnson, Joe Macintosh, and Geoff Marlowe. The 'Description' column is currently empty.

Name	Description
Henry Bush	
John Henderson	
Kathy Johnson	
Joe Macintosh	
Geoff Marlowe	

Figure 11 - Table as a result of an LDAP search on all disabled user accounts in a domain

#### Fixed table

The content of this table is either entered directly in UMRA or taken from a flat text file. It is used to to present the end user with specific fixed content in a form, such as a division, department, OU, domain etc. The characteristics of a fixed table are that the information is not derived from Active Directory, that it only contains 1 column and that it always has the same contents.



The image shows a 'Form preview' window with a title bar containing a yellow icon, the text 'Form preview', and standard window controls. The main content area is titled 'Select the department:' and contains a table with a single column. The table has three rows of data, each preceded by a blue globe icon. The entries are IT, Marketing (which is highlighted with a blue background), and Sales. The table has a scrollbar on the right side.

IT
Marketing
Sales

Figure 7 - Simple fixed table (list of entries)

### *Generic table - Database query*

In a large enterprise environment, not all user resource data are stored in Active Directory. Other information systems, an HR system or a phone system for example, may also hold user related information. In an UMRA project form, a generic table can be inserted to display information from these systems. As part of the table setup, the administrator needs to specify the database to connect to, as well as a database query to define which data should be retrieved and shown in the table.

The generic table for databases can be used to connect to **any** database or information system. MS Access databases can be accessed directly through the Jet engine. Other databases (SQL, ERP, HR systems, PeopleSoft, etc.) can be accessed through OLE DB. The number of available OLE DB providers will depend on your local configuration (see Figure 12).



*Figure 12 - List of available OLE DB database connections*

The option to connect to other databases is an extremely powerful feature in UMRA, since it allows you to combine Active Directory with user data from a wide range of other information systems. Figures 9 and 10 show an example where a generic form table is configured to connect to an HR system (SQL server). When the connection settings have been specified correctly, the message “Test connection succeeded” will be displayed when you click the **Test Connection** button.

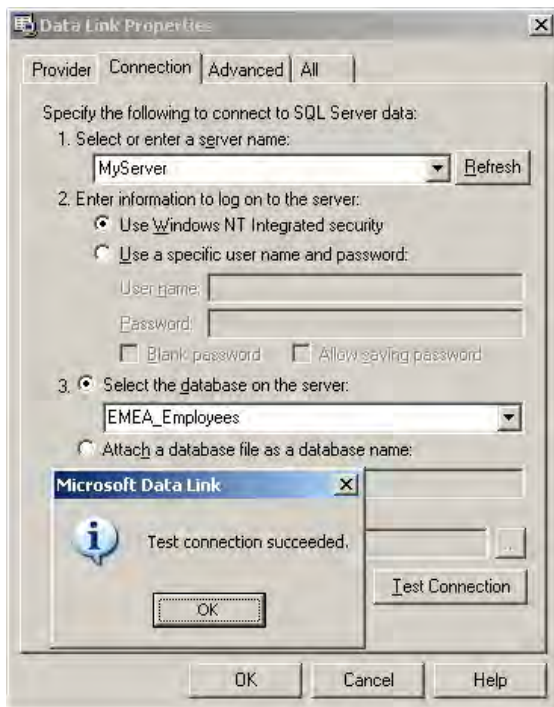
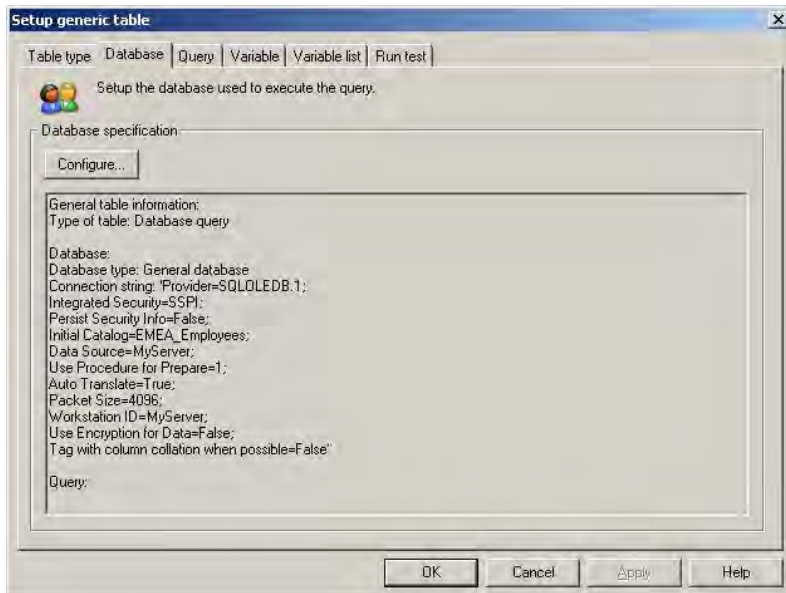


Figure 13 - Setting up a SQL database connection

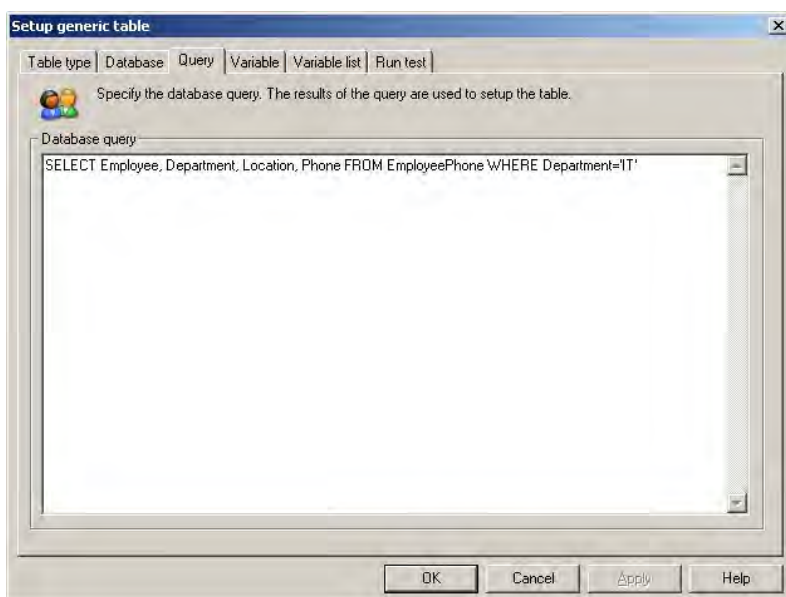
When the connection data have been specified, a binding string is created to connect to the database.



*Figure 14 - Binding string to connect to the database*

Once the connection has been properly set up, a SQL query can be created. In the example in figure 15, a SQL query has been specified to retrieve names of employees, locations and phone numbers for the IT department from the **EmployeePhone** table:

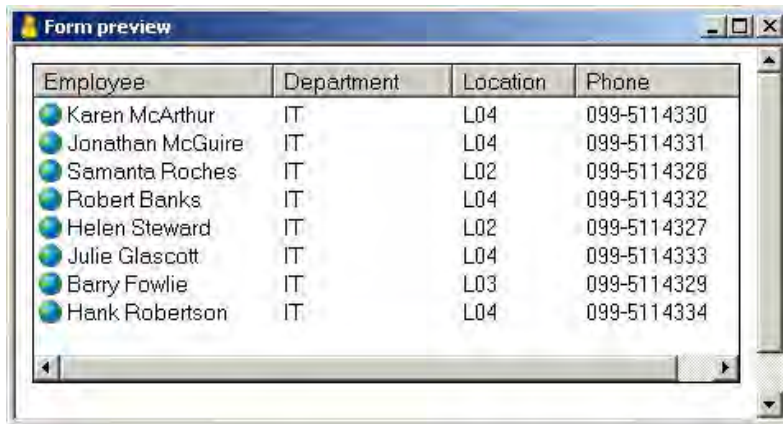
**SELECT Employee, Department, Location, Phone FROM EmployeePhone WHERE Department='IT'.**



*Figure 15 - Specifying the database query*



The result of this database query is shown in figure 16. The query has returned the data in the columns Employee, Department, Location and Phone for all rows where Department is equal to "IT".

A screenshot of a 'Form preview' window. It contains a table with four columns: Employee, Department, Location, and Phone. The table lists eight employees, all of whom are in the IT department. Each employee name is preceded by a small globe icon. The window has a standard Windows-style title bar and a scrollbar on the right side.

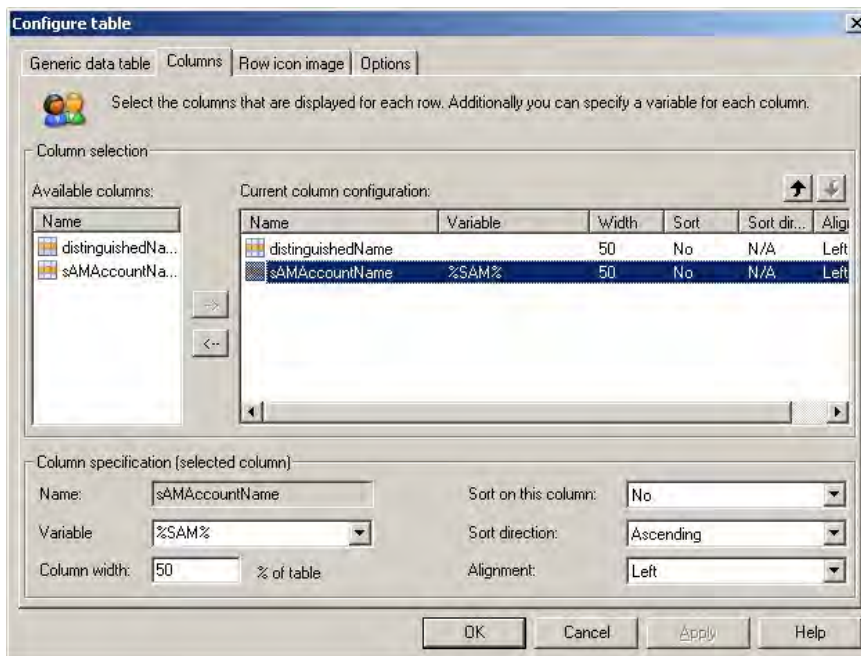
Employee	Department	Location	Phone
Karen McArthur	IT	L04	099-5114330
Jonathan McGuire	IT	L04	099-5114331
Samanta Roches	IT	L02	099-5114328
Robert Banks	IT	L04	099-5114332
Helen Steward	IT	L02	099-5114327
Julie Glascott	IT	L04	099-5114333
Barry Fowlie	IT	L03	099-5114329
Hank Robertson	IT	L04	099-5114334

*Figure 16 - Result of a database query is shown in an UMRA form*



### Specifying columns

For each of the before mentioned generic tables you need to specify which columns of the table you would like to display. This can be configured in the **Column** tab of the **Configure table** window (see figure below). In the same window, you can specify the sort order for the selected column and assign a variable to the selected column.



For the generic table **Variable** you first need to specify the column headers, which is discussed in *Special table type - Generic table Variable* on page 16.

### 3.8.3. Special table type - Generic table Variable

Not all user relevant data can be captured using an LDAP or database query. Table data are also generated by some specific script actions, in which case the collected data are stored in a variable, **instead of being generated as part of a form table object**. UMRA comes with several built-in script actions producing table output, stored in a variable:

**List services status** – script action to manage services;

**List printer documents** – script action to manage printers and printer queues;

**Generate generic table** - script action to generate a generic table (LDAP query, database query or variable) and store the result in a variable for further processing;

**LDAP script actions** to access other (non-Active Directory) directory service like Novell eDirectory and Linux OpenLDAP;

**Manage table data** – script action for creating, editing, and merging tables. See section *Programmatically creating and evaluating tables* on page 20 and the **Help** for more information.

Projects making use of these script actions, usually consist of two parts. The first project, or “auxiliary” project can be defined as the project in which the project data are collected using one of the above mentioned script actions. The result is stored in a variable. In the second project, or “main” project, a table of the variable type can be inserted to show the contents of this variable. Script actions can then be executed for selected rows in the table (see the figure below).

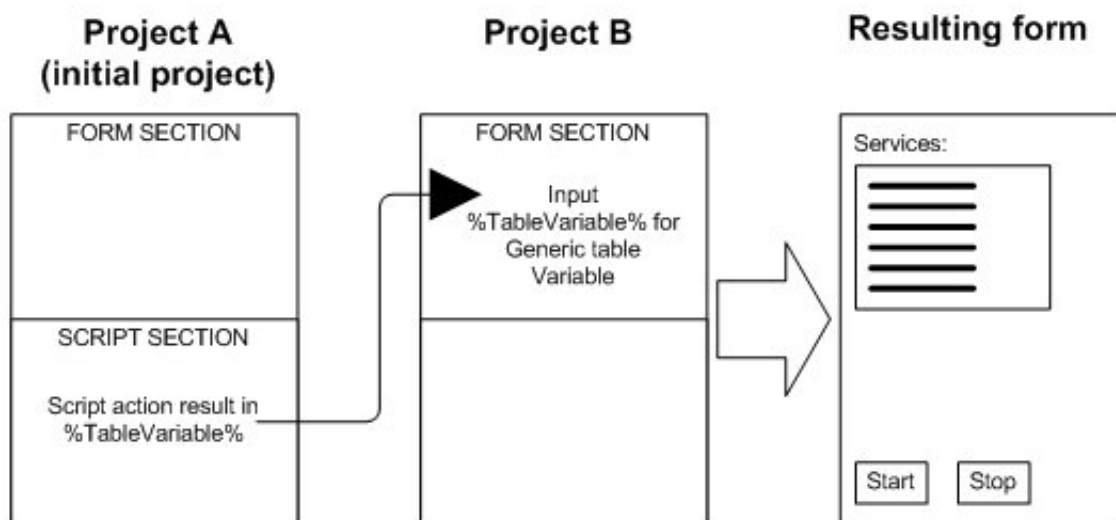
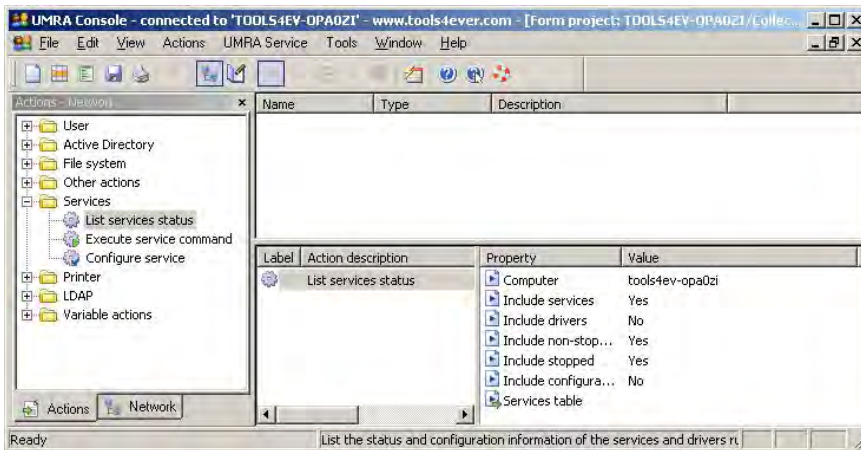


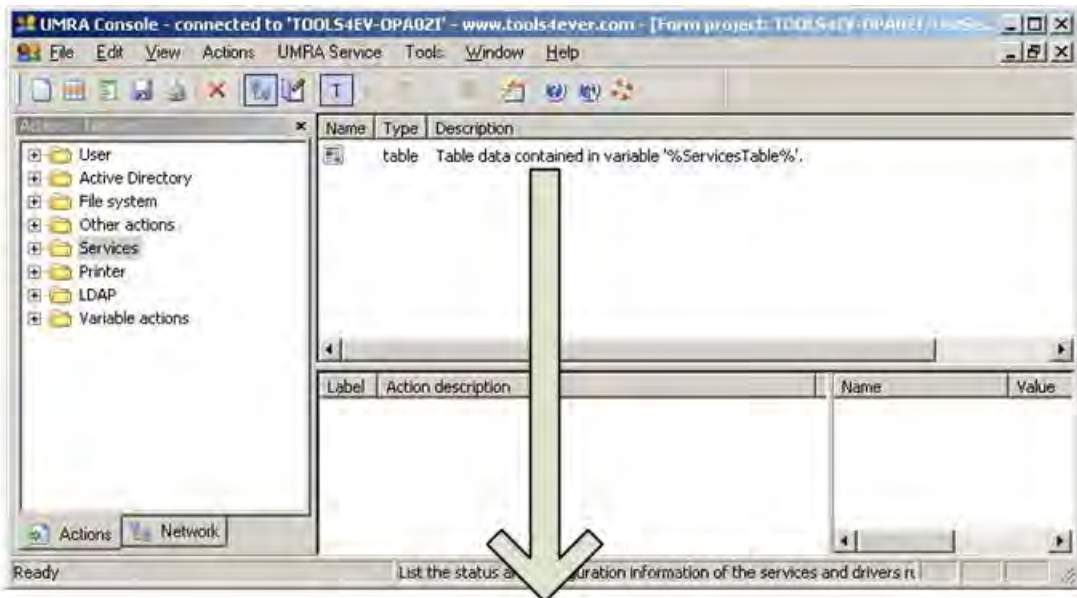
Figure 17 - Concept of the generic table Variable

The practical use of this powerful concept is illustrated in figure 17. Project A contains the script action “List services status”. The result of this script action is stored in table format in a variable called **%ServicesTable%**.



*Figure 18 - Project A - Running a script action collecting data in table format. The results are stored in a variable.*

By defining project A as an initial project in project B, the script in project A will be executed before the form of project B is displayed. The variables in project A will be passed to project B. In project B (see Figure 15), a generic table of the variable type can now be inserted in the form window to display content of the variable **%ServicesTable%**.



**Form preview**

Computer	Internal name	Name	Status (text)	Status...	Process ID	Svc type (text)	Svc type...	Interactive
MyServer	NetDDE	Network...	stopped	1	0	service (shared ...	32	No
MyServer	NetDDEds...	Network...	stopped	1	0	service (shared ...	32	No
MyServer	Alerter	Alerter	stopped	1	0	service (shared ...	32	No
MyServer	Netlogon	Net Log...	running	4	852	service (shared ...	32	No
MyServer	AppMgmt	Applicat...	stopped	1	0	service (shared ...	32	No
MyServer	Netman	Network...	running	4	1460	service (shared ...	32	No
MyServer	AudioSrv	Window...	stopped	1	0	service (shared ...	32	No
MyServer	Nla	Network...	running	4	1460	service (shared ...	32	No
MyServer	BackupExe...	Backup ...	running	4	1184	service	16	No
MyServer	NtntSvc	Network...	stopped	1	0	service (shared ...	32	No
MyServer	BackupExe...	Backup ...	running	4	4020	service	16	No
MyServer	NSCTOP	Symant...	running	4	1912	service	16	No
MyServer	BackupExe...	Backup ...	running	4	3572	service	16	No
MyServer	Nlfrs	File Re...	running	4	224	service	16	No
MyServer	Browser	Comput...	running	4	1460	service (shared ...	32	No
MyServer	NtLmSsp	NT LM ...	running	4	852	service (shared ...	32	No
MyServer	ClipSrv	ClipBook	stopped	1	0	service	16	No
MyServer	NtmsSvc	Remov...	stopped	1	0	service (shared ...	32	No
MyServer	CryptSvc	Cryptog...	running	4	1460	service (shared ...	32	No
MyServer	PlugPlay	Plug an...	running	4	840	service (shared ...	32	No

*Figure 19 - Project B - The content of the variable is now displayed using the generic table Variable*

### Specifying columns for table type Variable

The special table type Variable takes its input from a variable containing table data. This variable (which is the result of script actions such as Generate Generic table, List services status, Get User Table) does not contain any header info. This means that you need to specify the correct column names. For variables which are the result of one of the before mentioned script actions, this is simply a matter of

selecting the right column template.

Script action	Variable	Use column template
Get user table	%UsersTable%	User info
List services status	%ServicesTable%	Services status (without config info) Services status (with config info). If you choose this template, 3 column names will be added.
List printer documents	%DocumentsTable%	Printer documents
List files and/or directories	User defined	Files and or directories list

Once you have defined the column names as indicated above, you can select the columns to be displayed and assign variables to columns using the **Columns** tab. This is done in the same way as for the other generic tables.

#### Generate Generic table - Script

It is also possible to perform LDAP queries and database queries as part of a script action and have the result stored as a variable. This is done using the script action **Generate generic table**. The principle of operation is identical to the one described above.

#### Programmatically creating and evaluating tables

Using the tables described in this document, you should be able to manage table data for most user management tasks. There are situations however, where these standard tables may not be sufficient:

- You wish to evaluate an existing table programmatically and / or create a new table from scratch ;
- You need to combine data from the tables described earlier, with data which are not contained in a table.

In such cases, you can use the For-Each and Manage table data script actions to create your own tables.

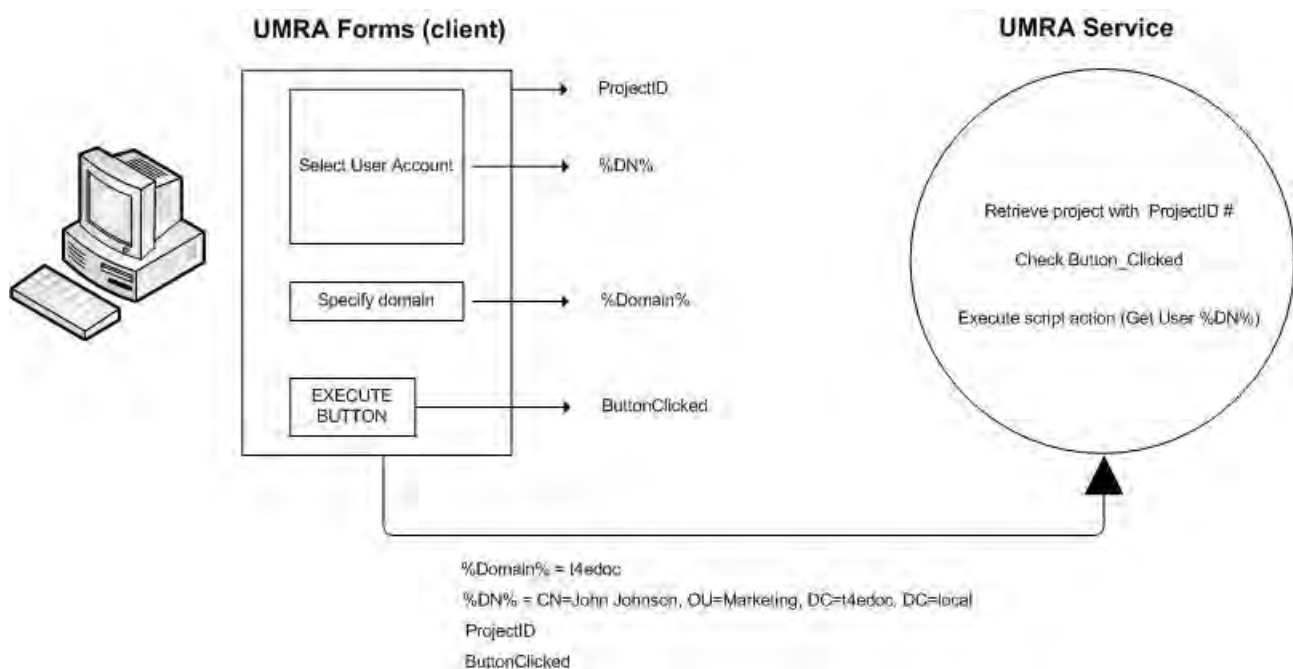
#### 3.8.4. Processing user input

When your form project includes a table from which the (Helpdesk) user can make a selection, the selected data will have to be processed. To understand how this works, we need to have a closer look at UMRA.

In the UMRA architecture, the form project as presented to the delegate user in **UMRA Form (client)** has been separated from the actual script. In other words, the UMRA project as shown on the client side does not contain any scripting. The project script, developed by the administrator in the **UMRA Console**, is part of the project maintained by the **UMRA Service**. The script actions in this project script make use of variables. As soon as a delegate user hits an action button in the project form on the client side, the following data are submitted to the **UMRA service**:

- the project form ID
- the name of the variables and their corresponding values
- the ID of the button which has been clicked

The **UMRA Service** then retrieves the project with the corresponding project ID from the forms database and executes the script of the project, substituting the variables in the script actions with the actual values. This principle is shown in figure 20.



*Figure 20 - Submitting variable values to the UMRA Service*

In the case of form tables, table columns can be assigned to a variable. This could be a table column holding the **distinguishedName** attribute as part of an LDAP table (see figure 21).

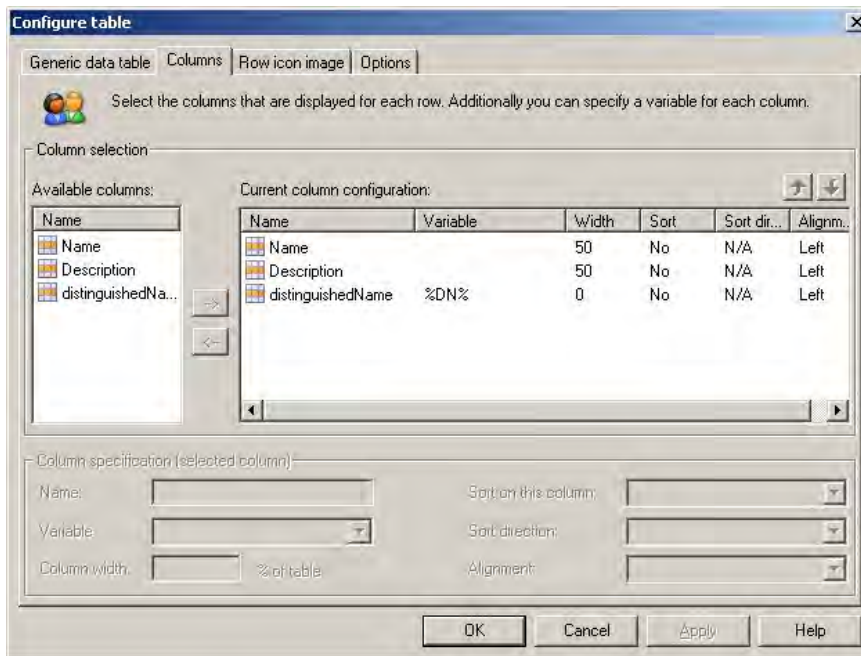


Figure 21 - Assigning variables to table columns

The **distinguishedName** attribute uniquely identifies users with the full LDAP string. For example:

**CN=John Johnson,OU=Marketing,DC=T4EDOC, DC=LOCAL**

When the user hits the action button in the form, the variable %DN% for the selected user ( e.g "CN=John Johnson,OU=Marketing,DC=T4EDOC, DC=LOCAL") is submitted to the **UMRA Service**. The UMRA Service then executes the script (e.g. **Get User (AD)**) of the project, substituting the variable %DN% in the script action property LDAP name with its actual value (see figure 22).

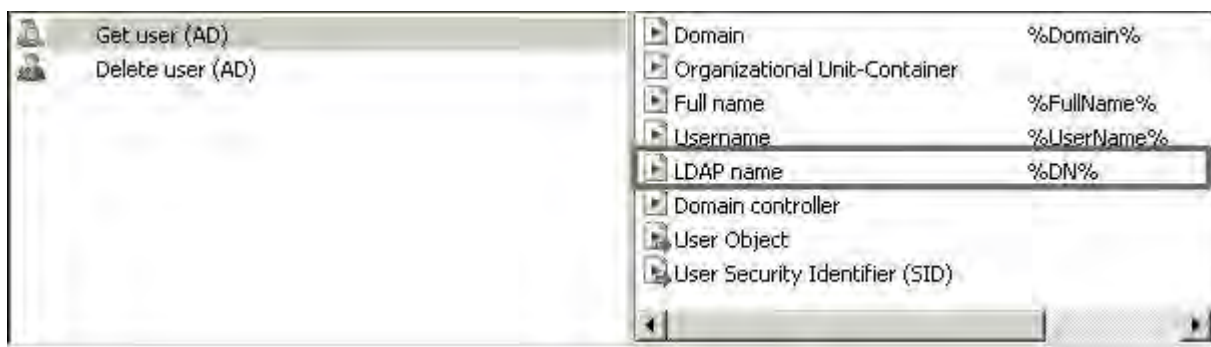


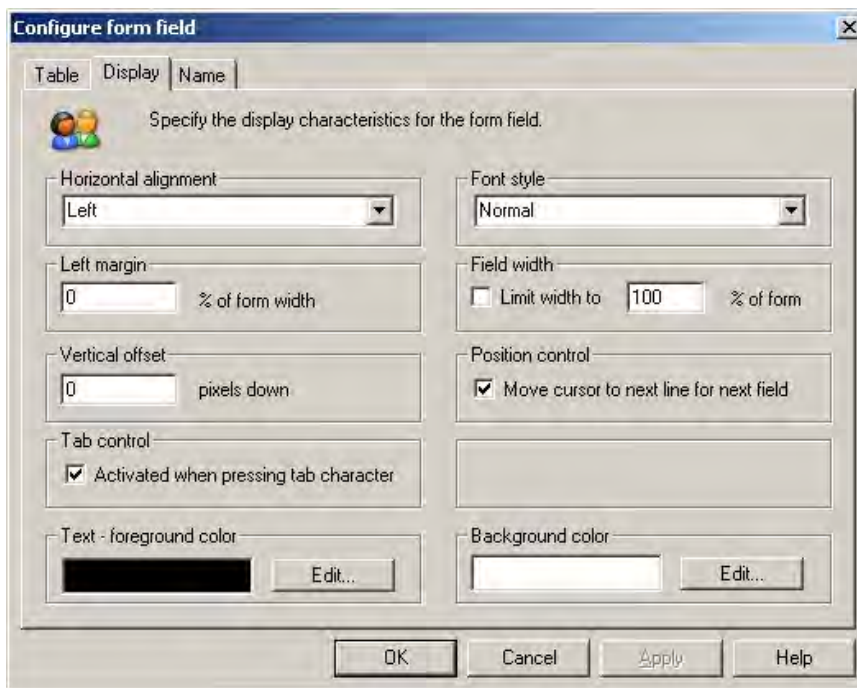
Figure 22 - Passing a variable holding the distinguishedName attribute to the Get user script action



### 3.8.5. Formatting tables

In UMRA, the display of tables can be fully customized. The table lay-out can be fully configured in the Configure form field window for tables (**Display** tab).

- Table alignment
- Margins
- Vertical offset
- Fonts
- Background colour
- Etc.





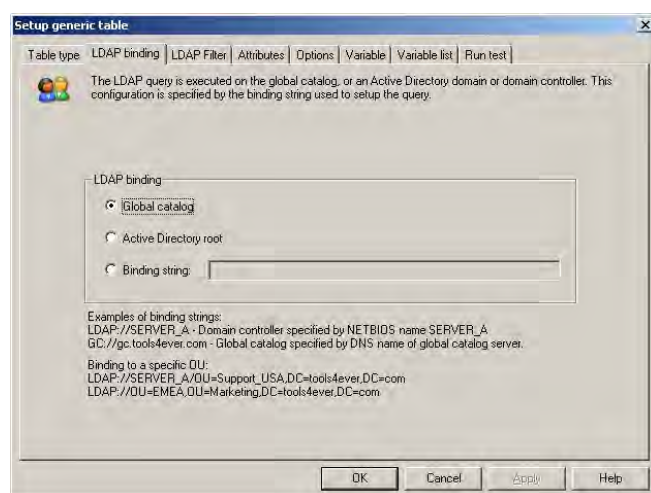
### 3.8.6. Using tables in UMRA - Forms & Delegation - Hands-on

#### Example 1 - Creating an LDAP table showing all disabled users in a domain

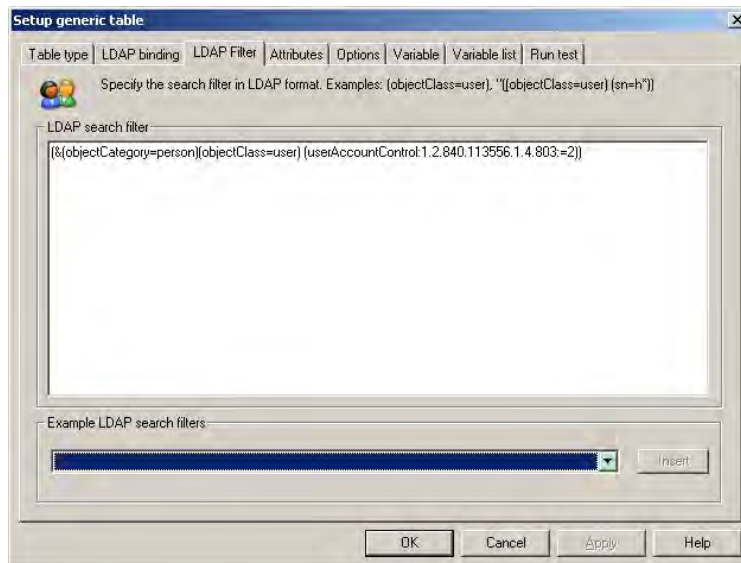
The project created in this example can also be found in the directory `\Tools4ever\User Management Resource Administrator\Example Projects\Forms\DisabledUsers.ufp`.

#### LDAP table - Creating a table listing all disabled users in a domain

1. Start the UMRA Console application and connect to the UMRA Service: Select **UMRA Service**, **Connect...** and connect to the computer on which the UMRA Service is installed.
2. Start the **UMRA Console** and create a new Forms project.
3. Right-click in the **Forms** window and choose the **Add form field** command.
4. Select the **Table** option.
5. Select the **Generic table** option and click the **Configure** button.
6. Click the **Configure** button once more.
7. Select the option **LDAP query** under **Table type**. A dialog box appears to configure your LDAP table.
8. Click the **LDAP binding** tab. In this window, the binding method for the LDAP query is specified. You can either choose a default binding to the Global Catalog or Active Directory root, or define your own binding string. For this example, we want to perform an LDAP search on all user objects in a domain, so we can select the option **Global Catalog**. This is a searchable master index with data about all objects in the domain.



9. Click the LDAP Filter tab to define a search filter.

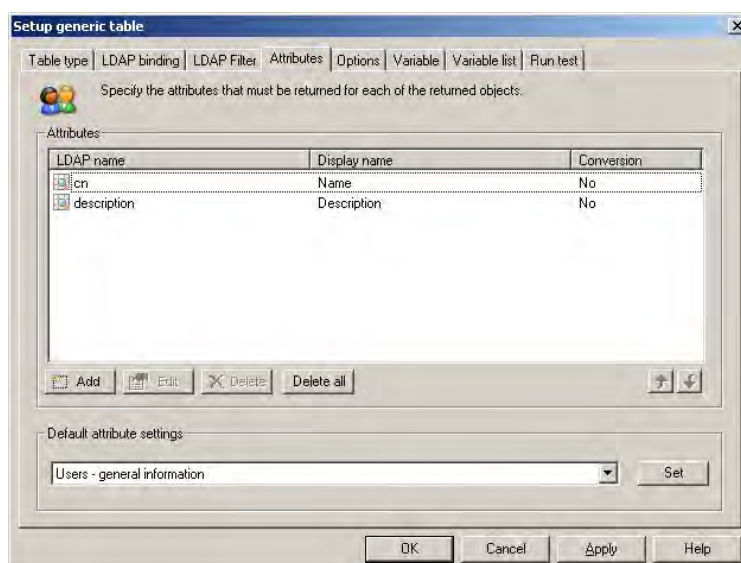


You can either enter an LDAP search query directly in the **LDAP search filter** window or select a predefined search filter from the **Example LDAP search filters** list box. For this example, please enter the following LDAP query:

**(&(objectCategory=person)(objectClass=user)  
(userAccountControl:1.2.840.113556.1.4.803:=2)).**

The next step is to specify which attributes should be returned.

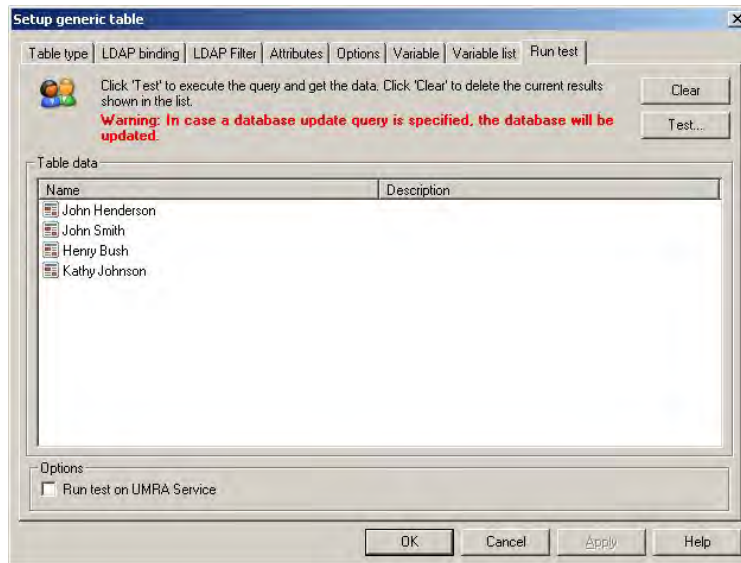
10. Click the **Attributes** tab.



The list of attributes for an Active Directory is endless. Which one you need to choose, also depends on the script action you wish to perform on the user selection. For this simple example, select the predefined attribute setting “Users – general information”. This will return the **cn**

attribute and the **description** attribute. Click the **Set** button when you are done. The LDAP query is now ready to be tested.

11. Click the **Run test** tab.



Click the **Test** button. The result of your LDAP query will appear in the **Table data** section. Click **OK** three times to return to the Forms window.

12. Right-click in the form window and choose the **Toggle auto preview** command to preview your table. The result should be similar to the figure shown below (the user names in your network will of course be different).

Name	Description
John Johnson	
krbtgt	Key Distribution Center Service Account
Henry Bush	
John Henderson	
Kathy Johnson	

You have now created an LDAP table which returns all the disabled users in a domain. Script actions can now be added to your project specifying what action needs to be performed on the selected data.

### Example 2 - Creating a form table to connect to a database

*As part of a delegation project, you want to show a delegate user a table containing the phone numbers of employees in the IT department. These data are available in the database EmployeePhoneNrs.mdb. In the example below we will show you how a table can be created in UMRA to display these data.*

The project created in this section can also be found in the directory **\Tools4ever\User Management Resource Administrator\Example Projects\Forms\GetPhoneNumbers.ufp**.

### LDAP table - Linking UMRA to an MS-Access database containing phone numbers for all employees, listed by department

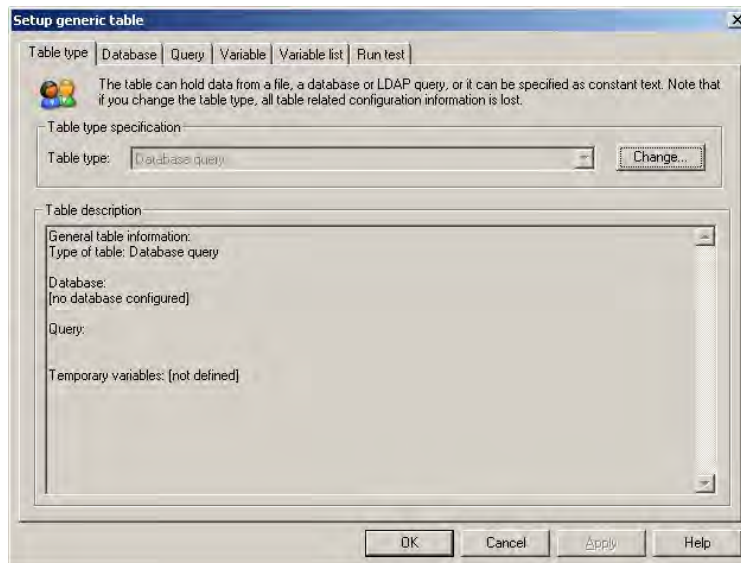
The MS Access database we need to link to, contains one table which is called **EmployeePhone**. This table holds the columns **ID**, **PersNr**, **Employee**, **Department**, **Manager**, **Location**, **FirstName** and **Phone**:

EmployeePhone : Table								
	ID	PersNr	Employee	Department	Manager	Location	FirstName	Phone
▶	1	1001	Henry Ford	Sales	Jonathan Garland	F10	Henry	099-5114919
	2	1002	Jonathan Garland	Sales		F10	Jonathan	099-5114920
	3	1003	John Johnson	Support	Linda Bard	F01	John	099-5114341
	4	1004	Donald Smith	Support		F02	Donald	099-5114342
	5	1005	Linda Bard	Support	Donald Smith	F03	John	099-5114343
	6	1006	Peter Jones	HR		K01	Peter	099-5114261
	7	1007	Rodney Briggs	HR		K01	Rodney	099-5114262
	8	1008	Jeff Hemingway	HR		E72	Jeff	099-5114300
	29	1009	Susan Woth	Helpdesk		L01	Susan	099-5114325
	30	1010	Brett Hewitt	Helpdesk		L01	Brett	099-5114326
	31	1011	Helen Steward	IT		L02	Helen	099-5114327
	32	1012	Samanta Roches	IT		L02	Samantha	099-5114328
	33	1013	Barry Fowlie	IT		L03	Barry	099-5114329
	34	1014	Karen McArthur	IT		L04	Karen	099-5114330
	35	1015	Jonathan McGuire	IT		L04	Jonathan	099-5114331
	36	1016	Robert Banks	IT		L04	Robert	099-5114332
	37	1017	Julie Glascott	IT		L04	Julie	099-5114333
	38	1018	Hank Robertson	IT		L04	Hank	099-5114334

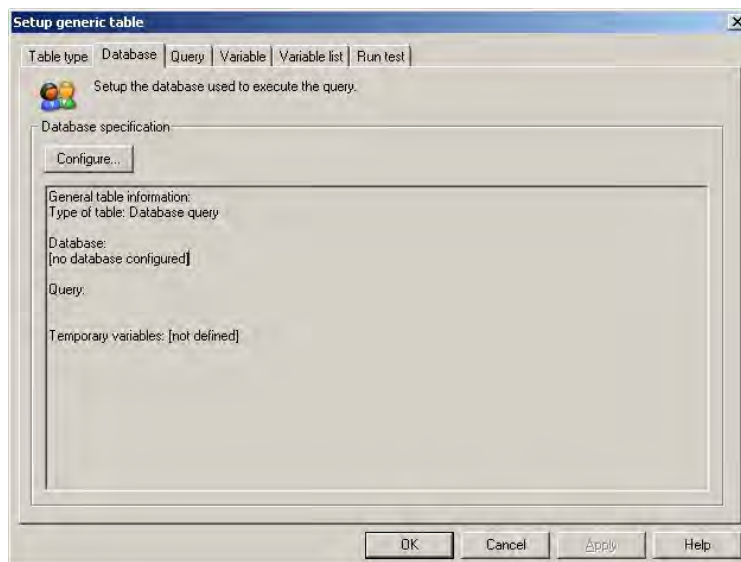
The link to the MS Access database is established through the Jet Engine, so there is no need to have MSAccess installed. In case you wish to explore the database yourself, you will find it in  
 \Tools4ever\User Management Resource Administrator\Example  
 Projects\Forms\EmployeePhoneNrs.mdb.

1. Start the **UMRA Console** application and connect to the **UMRA Service**: Select **UMRA Service**, **Connect...** and connect to the computer on which the **UMRA Service** is installed.
2. Create a new Forms project.
3. Right-click in the Forms window and choose the **Add form field** command.
4. Select the **Table** option.
5. Select the **Generic table** option and click the **Configure** button.
6. Click the **Configure** button once more.

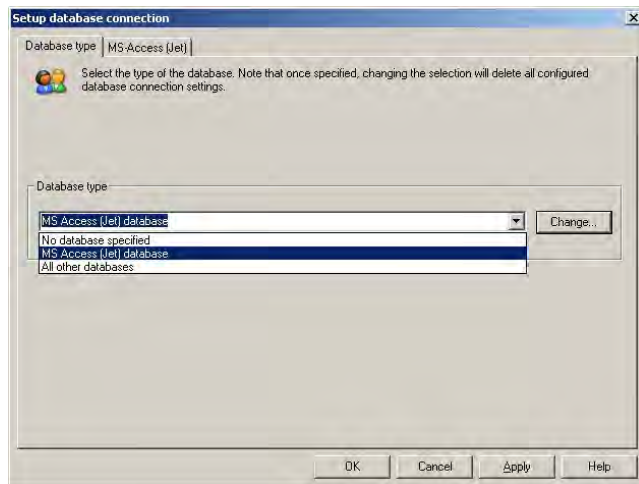
7. Select the option **Database query** under **Table type**. A dialog box appears in which you can configure your database table. The following step is to define the database type and name.



8. Click the **Database** tab. The following dialog box will appear:

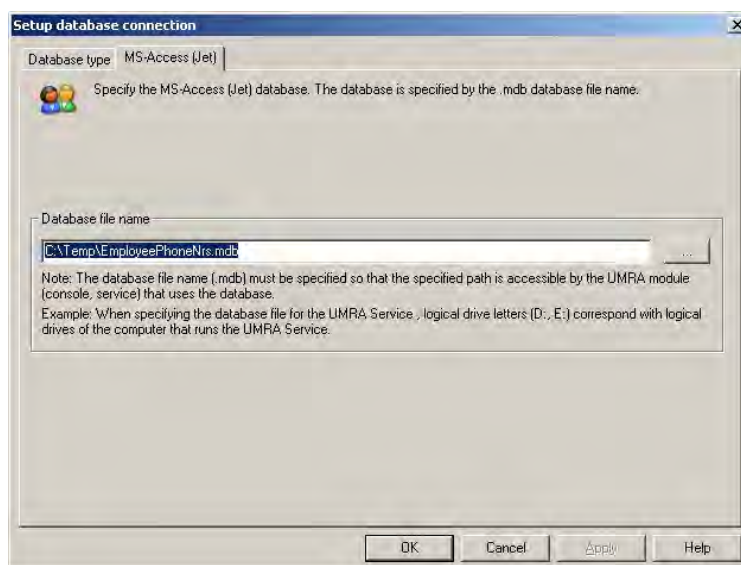


- Click the **Configure** button to specify the database you wish to use. Select the option **MS Access (Jet) database** from the **Database type** list.



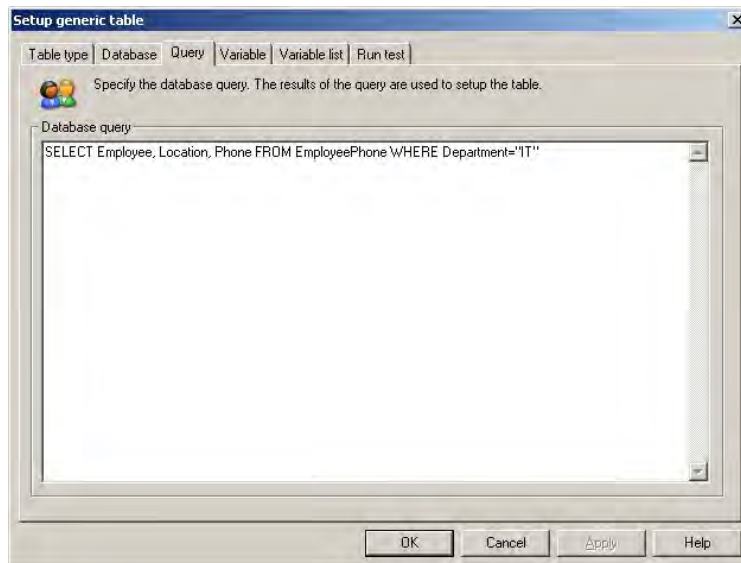
Note - linking to a database is not limited to an MS Access database. You can connect to any database for which a OLE DB provider is available. If your database is not included in the standard OLE DB list of OLE DB providers, please check with your database provider.

- Click the **MS-Access (Jet)** tab and browse to the file **EmployeePhoneNrs.mdb**. Click the **Open** button.

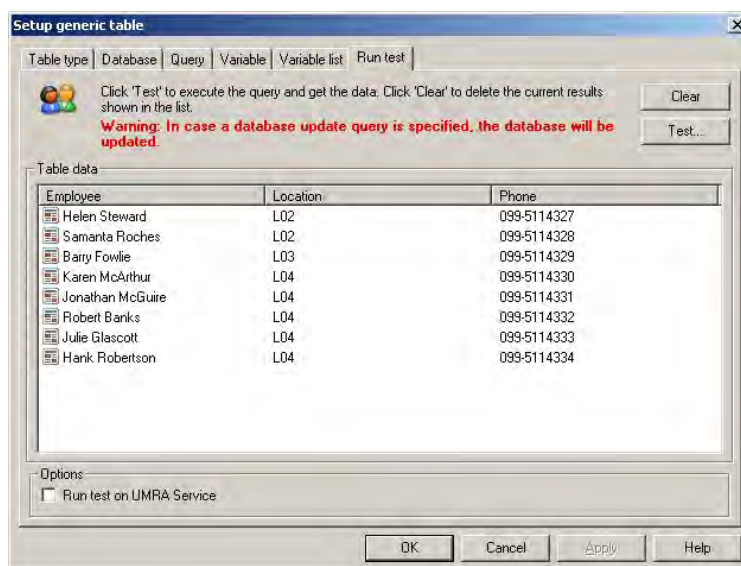




11. Click **OK** to return to the database setup window. We have now specified which database needs to be used. In order to specify which data we want to retrieve from the database, a database query must be specified. Click the **Query** tab. The following dialog box will appear:



12. Enter the following query:
13. **SELECT Employee, Location, Phone FROM EmployeePhone WHERE Departments="IT"**
14. This query will return all records in the columns Employee, Location and Phone of the **EmployeePhone** table (the table name you obtained in step 2) where the column Department is "IT".
15. Click the **Run test** tab and click the Test button. The following data should appear:



In the **Columns** tab, the displayed columns can be changed.

16. Click the **Columns** tab.

This window is used to configure which columns must be shown in the form. Here you also specify the variables that are passed to the UMRA Service when the end-user selects a table entry and presses a submit button. On the left side, the **Available columns** are shown. When the **Run test** was performed successfully in step 13, the actual column names will be shown here. Change the column width for column 1-3 as shown in figure 23. Exclude the columns 4-10 by selecting the column and clicking the left arrow (←). Finally, click **OK**.

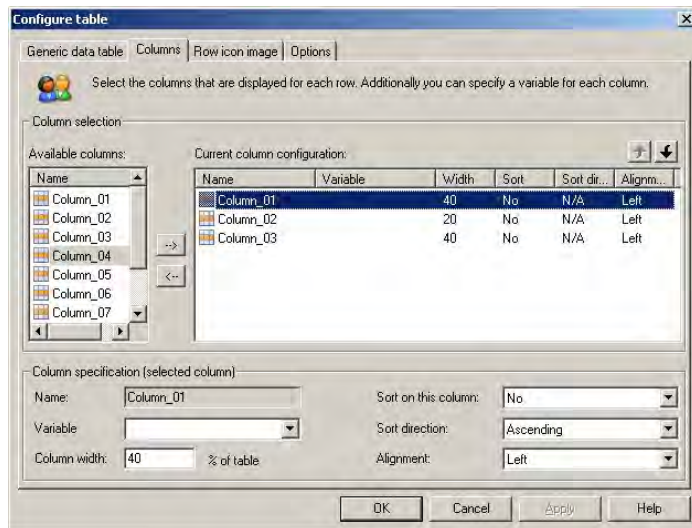


Figure 23 - Configuring the columns to be displayed in the form table

17. Click **OK**. When you run the preview, the resulting table as shown in figure 24 will be displayed.

Employee	Location	Phone
Karen McArthur	L04	099-5114330
Jonathan McGuire	L04	099-5114331
Samanta Roches	L02	099-5114328
Robert Banks	L04	099-5114332
Helen Steward	L02	099-5114327
Julie Glascott	L04	099-5114333
Barry Fowlie	L03	099-5114329
Hank Robertson	L04	099-5114334

Figure 24 - Displaying database data in a form table

You have now successfully created a form table object to hold database content.



### Example 3 - Creating a variable with table data and showing the content in a form table

In this exercise, the project **Collect Services** is created which collects the services information of a specific computer. These data can subsequently be shown in another project defining how to manage these services. Assigning script actions to the table data in the second project falls outside the scope of this exercise, but is described in detail in the document “*UMRA Example projects: Service Management*”.

The project created in this example can also be found in the directory **\Tools4ever\User Management Resource Administrator\Example Projects\Forms\CollectServices.ufp** and **ShowServices.ufp**.

#### Project A - Collecting services

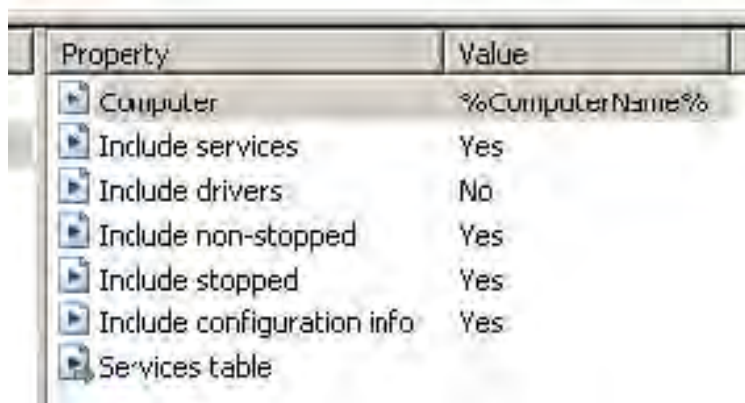
1. Start the **UMRA Console** application and connect to the **UMRA Service**: Select **UMRA Service, Connect...** and connect to the computer on which the **UMRA Service** is installed.
2. Choose **File→New**. Select the option **Form project** and click **OK**. Enter the name of the project, **Collect Services**, and click **OK**.
3. First we will set up the action that creates and initializes the variable holding the name of the computer of which we want to manage the services.
4. Drag the **Set variable** script action into the script action window and configure this script action as shown in figure 25. In this example, the name of the computer is “**SERVER\_A**”.



Figure 25 - Defining the variable for the computer name

Next, we need to collect the service status information for the computer “**SERVER\_A**”.

5. In the **Actions** bar, drag the script action **List services status** from the **Services** folder to the script section window. Specify the properties for this script action as shown in figure 26.



Property	Value
Computer	%ComputerName%
Include services	Yes
Include drivers	No
Include non-stopped	Yes
Include stopped	Yes
Include configuration info	Yes
Services table	

Figure 26 - Configuring the properties for the **List services status** script action

The UMRA software will connect to the computer specified by **%ComputerName%** and collect the status of all services. The status information includes the name of the service, the operational state of each service (running, stopped), type of service (automatic, manual, disabled) and so on. This information is stored as a table in the variable **%ServicesTable%**. In other words, this single variable will hold a table with multiple rows and columns. The content of this variable can be displayed as a form table in another project for managing the collected services.

Finally, the security settings for the project **Collect Services** must be specified.

6. Choose the **Form properties** command from the **Actions** menu and click the **Security** tab. For this exercise, you can set the group to "Everyone".
7. Save the project and close the project window

In the second project, a form table will be set up to display the content of the variable **%ServicesTable%**.

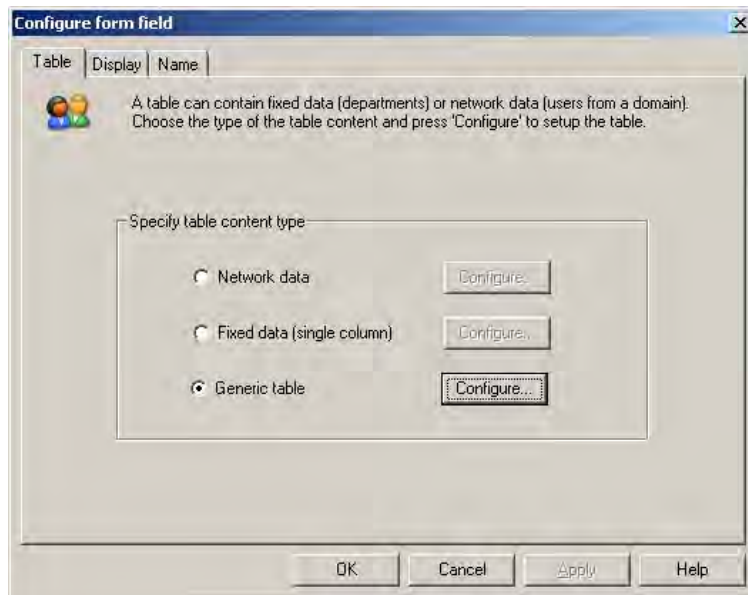
*Project B - Inserting a form table to display table content in a variable*

1. Choose **File→New**. Select the option **Form project** and click **OK**. Enter the project name, **Show Services**, and click **OK**.

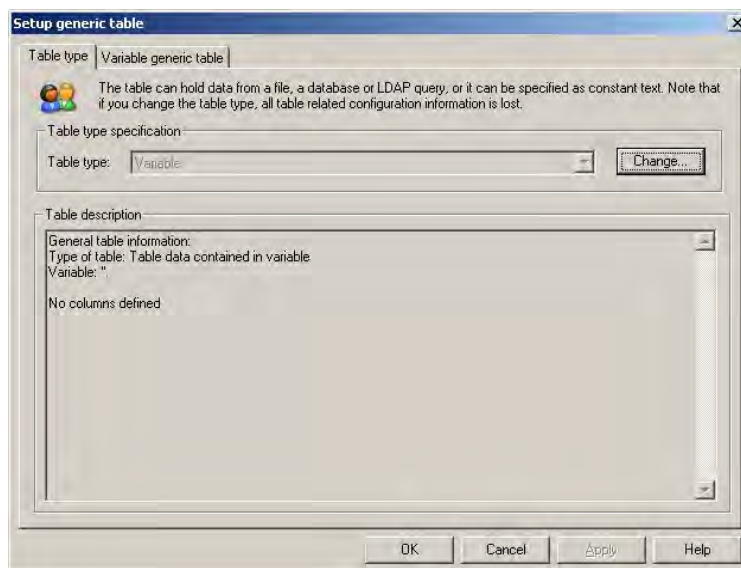
Next, the project **CollectServices** must be specified as an initial project to ensure that the variable **%ServicesTable%** is properly passed to the **Show Services** project.

2. Right-click the Form window and select **Form properties....** Select the **Initial project** tab and select **CollectServices** as the initial project. Click **OK**.

3. Insert a table object in the Forms window. Select the **Generic table** option and click the **Configure** button.



The table data we wish to use are contained in the variable %ServicesTable%, so we need to select the generic table type Variable as shown in the figure below.



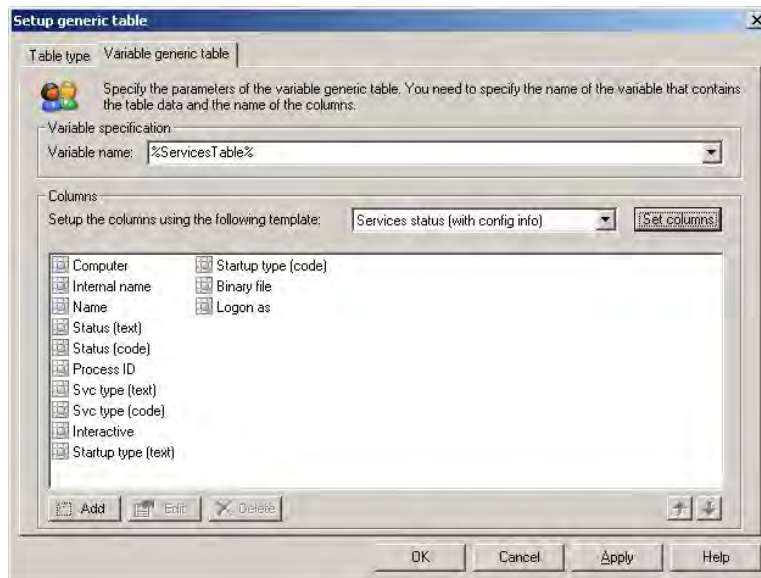
For this table type, the name of the variable and the columns contained in this variable need to be specified in the **Variable generic table** tab.

4. Enter %ServicesTable% in the Variable name list.

Specifying the column names is necessary because a table variable only holds the data of the table, not the column names. Built-in column templates are available for this purpose.

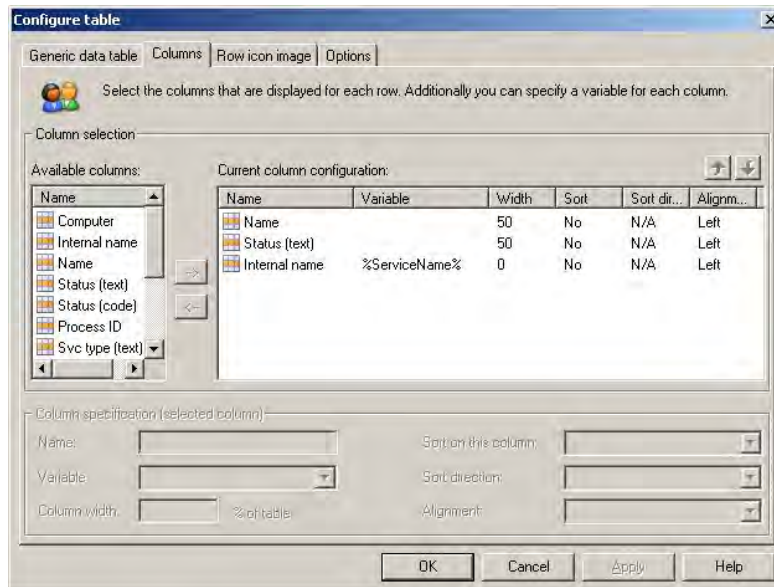
5. Select the column template **Services status (with config info)** in the **Columns** section and click the **Set columns** button. This will include the column names "Computer", "Internal name",

“Name”, “Service”, “Status (text)”, “Status (code)”, “Process ID”, “Svc type (text)”, “Svc type (code)”, “Interactive”, “Startup-type (text)”, “Startup-type (code)”, “Binary file” and “Logon as” |. See the Help for more detailed information regarding these status fields. Finally, click **OK**.



If you wish to make changes to the displayed columns, you can do so in the **Columns** tab.

6. Click the **Columns** tab.

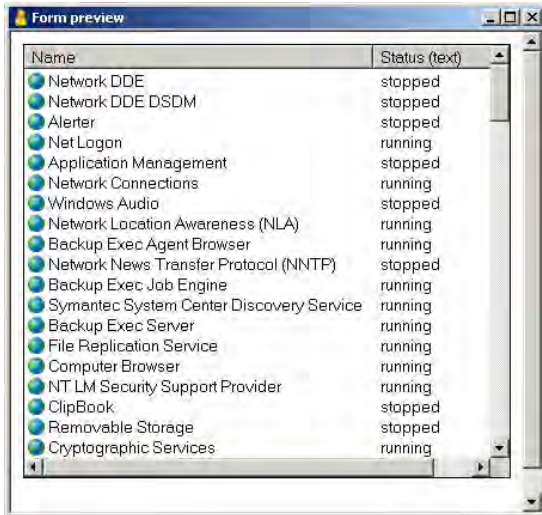


This window is used to configure which columns must be shown in the form. Here you also specify the variables that are passed to the UMRA Service when the end-user selects a service and presses a submit button. On the left hand side, the available columns are shown. These columns correspond with the columns configured in the previous step.

By using the add (->) and remove (<-) buttons you can set up and modify a column configuration. In the example shown, the form will have a table with 3 columns. The third column (Internal

name) will not be visible since it has a width of 0%. This column is included since it uniquely specifies the name of the service, but there is no need to display it for the end user. When the user selects a service and presses a button, the value of this column is stored in variable `%ServiceName%`. This variable is passed to the UMRA Service and used for further processing.

7. Click **OK**. When you run the preview, the resulting table as shown in the figure below.



The image shows a 'Form preview' window with a table of Windows services. The table has two columns: 'Name' and 'Status (text)'. The services listed are: Network DDE (stopped), Network DDE DSDM (stopped), Alert (stopped), Net Logon (running), Application Management (stopped), Network Connections (running), Windows Audio (stopped), Network Location Awareness (NLA) (running), Backup Exec Agent Browser (running), Network News Transfer Protocol (NNTP) (stopped), Backup Exec Job Engine (running), Symantec System Center Discovery Service (running), Backup Exec Server (running), File Replication Service (running), Computer Browser (running), NT LM Security Support Provider (running), ClipBook (stopped), Removable Storage (stopped), and Cryptographic Services (running).

Name	Status (text)
Network DDE	stopped
Network DDE DSDM	stopped
Alert	stopped
Net Logon	running
Application Management	stopped
Network Connections	running
Windows Audio	stopped
Network Location Awareness (NLA)	running
Backup Exec Agent Browser	running
Network News Transfer Protocol (NNTP)	stopped
Backup Exec Job Engine	running
Symantec System Center Discovery Service	running
Backup Exec Server	running
File Replication Service	running
Computer Browser	running
NT LM Security Support Provider	running
ClipBook	stopped
Removable Storage	stopped
Cryptographic Services	running

### 3.8.7. Contacts

You can visit our Tools4Ever web site for contact, support and other information about our products:

<http://www.tools4ever.com/> (<http://www.tools4ever.com/> <http://www.tools4ever.com/>)

<http://forum.tools4ever.com/> (<http://forum.tools4ever.com/> <http://forum.tools4ever.com/>)

## 3.9. Lotus Notes user guide

The topics in this section describe how to configure Umra for user with Lotus Notes

### 3.9.1. Configuring the UMRA console for use with Lotus Notes

This topic describes step by step how to configure **UMRA console application** for use with **Lotus Notes**. It assumes that you have already have successfully installed and configured the UMRA console and the UMRA Service for general usage. It assumes also a basic level of understanding of both Lotus Notes administration and of UMRA.

You need configure the UMRA *console* for use with Lotus Notes if you want to run local mass projects with the console that make use of the Lotus Notes functionality. For all other project types you need to configure the Umra *service*. See *Configuring the UMRA service for use with Lotus Notes* on page 42 for instructions.

#### Step by step configuration of the UMRA Console application for use with Lotus Notes.

1. Install the Lotus Notes client software.

Install the **Lotus Notes client** and the **Lotus Domino Admin** software of IBM on the computer that runs the **UMRA Console**, configure these applications, and verify their proper operation

2. Create a Lotus Notes initialization file for the UMRA console application.

When the UMRA console connects to the Lotus Notes environment it needs to provide a notes initialization file with all relevant connection parameters. Such a file can be created as follows:

- a) Log on to Windows with the account you use to run the UMRA console.
- b) Start the IBM **Lotus Notes client** and verify the correct operation of the application. Make sure

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

you connect to the **Domino server** with a **Lotus Notes user ID** that has sufficient administrative rights in Lotus Notes. It is this **Lotus Notes User ID** that will be used by the **UMRA console** to perform all its actions in Lotus Notes. You may consider using the **administrator account** of Lotus Notes for this purpose, or alternatively create a special Lotus Notes user with administrative rights.

- c) Exit the Lotus Notes client application.
  - d) Locate the Lotus Notes **Program directory**, and find the file **Notes.ini**. The default location is c:\Program Files\Lotus\notes\notes.ini.
  - e) Copy the file **Notes.ini** to a new file, e.g. **UMRAConsoleNotes.ini**.
3. Configure a password dll file.

In order to be able to exchange session passwords between the UMRA application and the Lotus Notes environment, Lotus Notes requires a special password dll file. The Lotus Notes software will access the password dll file (part of the UMRA software) to obtain a (session) password when it requires one. This is required to prevent the appearance of Lotus notes login screens at inconvenient moments, which may impair proper operation of the console.

- a) Locate the **UMRA program directory** (default c:\program files\Tools4ever\User Management Resource Administrator).
- b) This directory contains the file **UMInpw.dll**. Copy this file to the Lotus Notes program directory, (i.e the directory containing the UMRAConsoleNotes.ini file).
- c) Open a text editor, for instance Notepad.exe, to edit the file **UMRAConsoleNotes.ini**. Navigate to the end of the file and add an extra line:

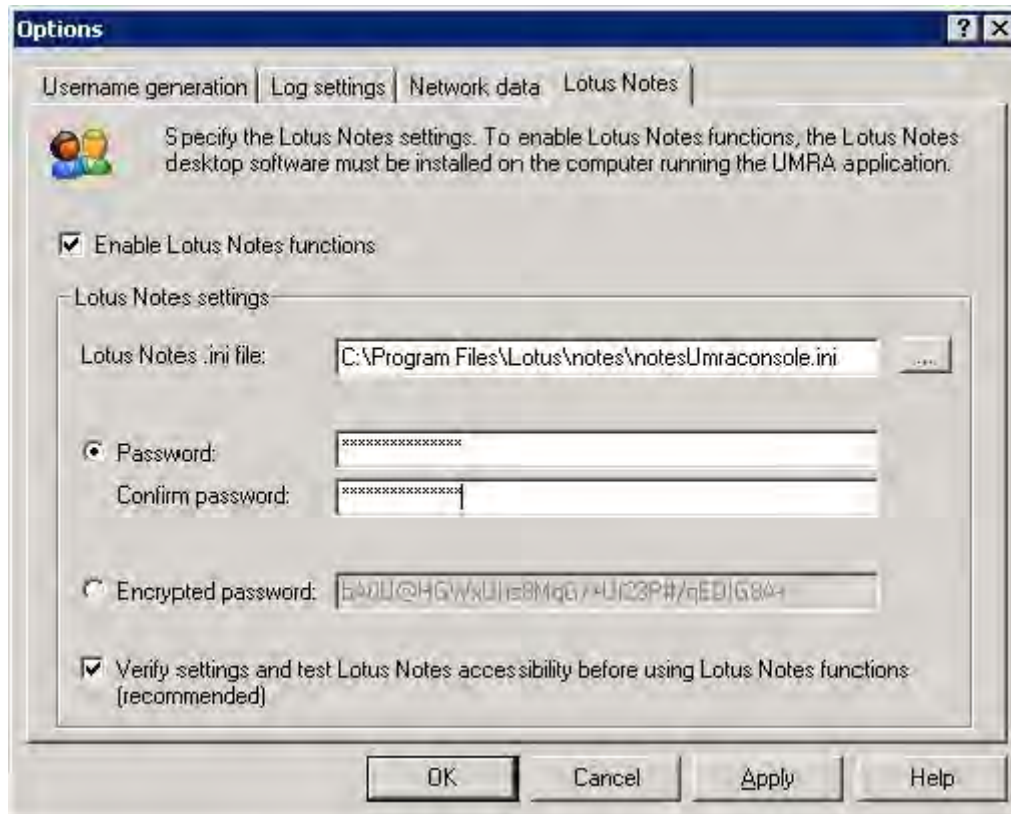
**EXTMGR\_ADDINS=UMInpw.dll** and make sure that there still is an **empty line** at the end of the file.

This configuration setting instructs the Lotus Notes software to access the specified dll (part of the UMRA software) to obtain a (session) password when it requires one.  
The resulting ini file will look something like this:

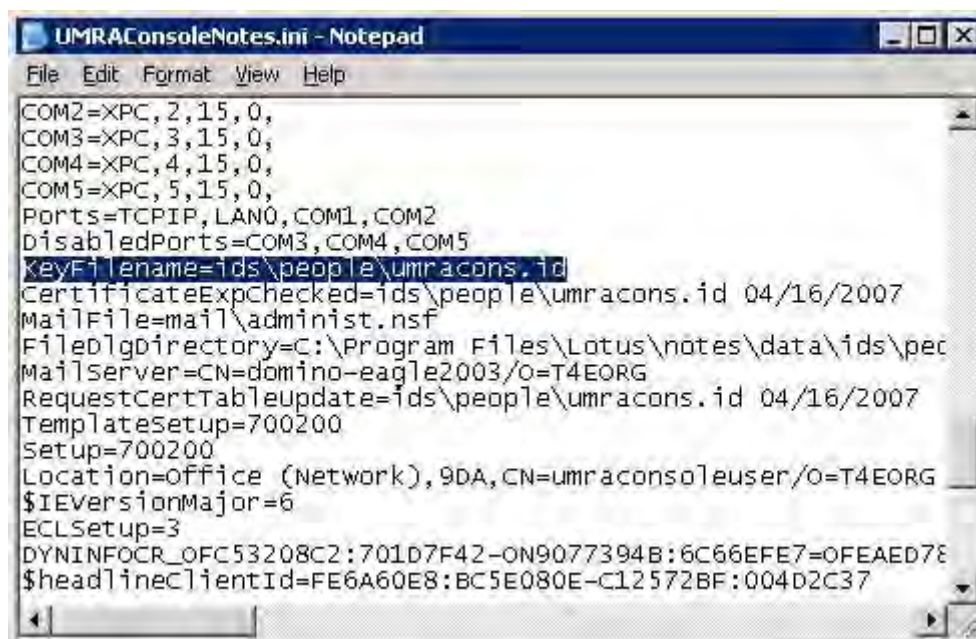








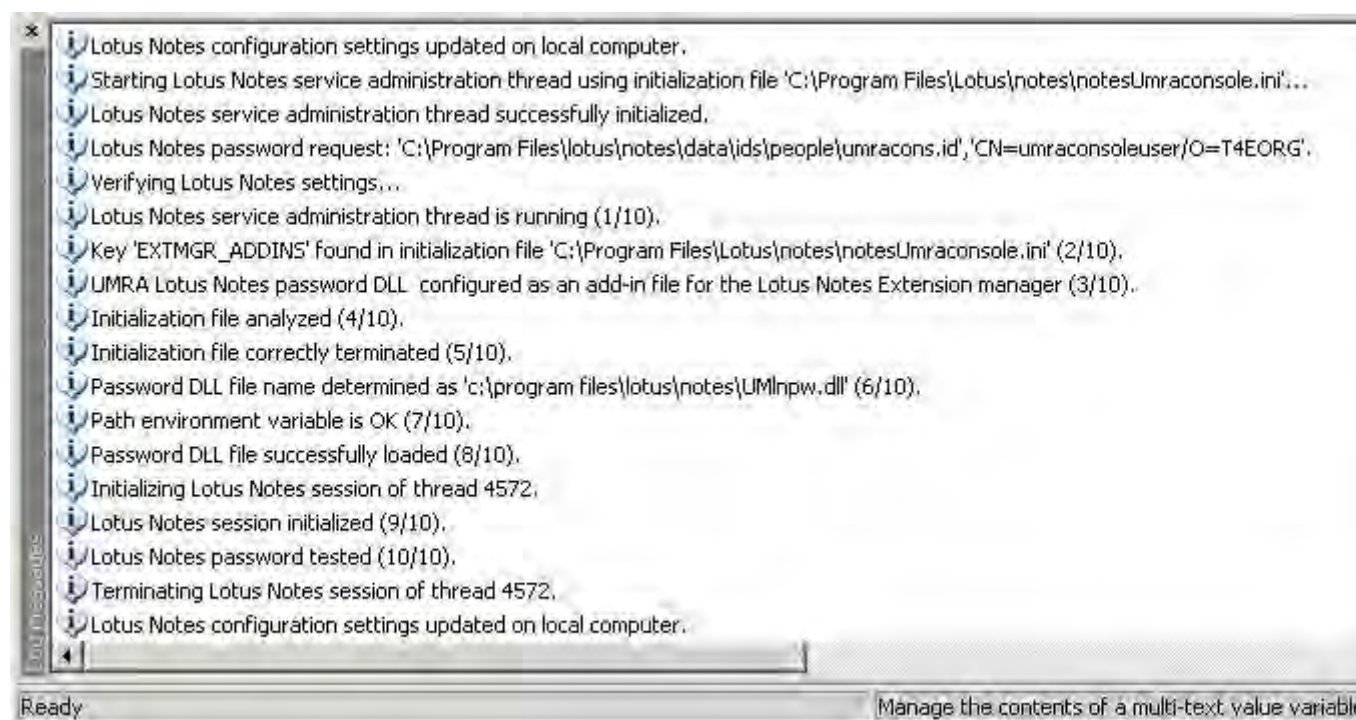
- c) Select the checkbox **Enable Lotus Notes functions**, and specify the Lotus Notes initialization file you have created in the previous steps. Specify the **password**. The password must be the password of the **Lotus Notes User ID file** that is listed in the **KeyFilename** property in the specified ini file. If you have followed the above procedure, this will be the Lotus Notes User ID file of the user you specified in the Lotus Notes Client.



```
UMRAConsoleNotes.ini - Notepad
File Edit Format View Help
COM2=XPC,2,15,0,
COM3=XPC,3,15,0,
COM4=XPC,4,15,0,
COM5=XPC,5,15,0,
Ports=TCPIP,LAN0,COM1,COM2
DisabledPorts=COM3,COM4,COM5
keyfilename=ids\people\umracons.id
CertificateExpChecked=ids\people\umracons.id 04/16/2007
MailFile=mail\administ.nsf
FileDlGDirectory=C:\Program Files\Lotus\notes\data\ids\pec
MailServer=CN=domino-eagle2003/o=T4EORG
RequestCertTableupdate=ids\people\umracons.id 04/16/2007
TemplateSetup=700200
Setup=700200
Location=Office (Network),9DA,CN=umraconsoleuser/o=T4EORG
$IEVersionMajor=6
ECLSetup=3
DYNINFOCR_OF53208C2:701D7F42-ON9077394B:6C66EFE7=OFEAED7E
$headlineClientId=FE6A60E8:BC5E080E-C12572BF:004D2C37
```

Note that the Lotus Notes client has created many items in the .ini file. Some are required for the UMRA connection, many others are not. However, unless you are very familiar with their meaning, we suggest not altering this file manually. If for some reason you want UMRA to connect to the Lotus Notes environment with a different user ID, please repeat the steps 2 and 3.

- d) Press OK to continue.
- e) If you now examine the **log messages** in the console, you should see something like:



- Now the Console is successfully initialized. You can start using the Lotus Notes actions in your local UMRA projects.

#### Configuration of the UMRA Service for use with Lotus Notes.

If you want to use the Lotus notes functionality of UMRA also when using service based projects, which is most likely, you also need to configure the UMRA service for use with Lotus Notes. The method of configuring the UMRA Service for use with Lotus Notes requires comparable steps as configuring the UMRA console. See *Configuring the UMRA service for use with Lotus Notes* on page 42 for a description how to do this.

#### 3.9.2. Configuring the UMRA service for use with Lotus Notes

This topic describes step by step how to configure **UMRA Service** application for use with Lotus Notes. It assumes that you have already have successfully installed and configured the UMRA console and the UMRA Service for general usage. It assumes also a basic level of understanding of both Lotus Notes administration and of UMRA.

You need configure the UMRA *console* for use with Lotus Notes if you want to run local mass projects with the console that make use of the Lotus Notes functionality. For all other project types you need to configure the Umra *service*. See *Configuring the UMRA Console for use with Lotus Notes* on page 37 for a description how to configure the console.

### Step by step Configuration of the UMRA Service for use with Lotus Notes

1. Install the **Lotus Notes client** and the **Lotus Domino Admin** program of IBM on the server that runs the UMRA Service. Configure these applications and verify their correct operation.
2. Create a Lotus Notes initialization file for the UMRA Service.

When the UMRA Service connects to the Lotus Notes environment it needs to provide a Lotus Notes initialization file with all relevant connection parameters. Such a file can be created as follows:

- a) Log on locally to Windows at the server running the UMRA service, with the same Windows account that is used by the UMRA service application itself.
  - b) Start the IBM **Lotus Notes client** and verify the correct operation of the application. Make sure you connect to the **Domino server** with a **Lotus Notes user ID** that has sufficient administrative rights in Lotus Notes. It is this **Lotus Notes User ID** that will be used by the **UMRA service** to perform all its actions in Lotus Notes. You may consider using the **administrator account** of Lotus Notes for this purpose, or alternatively create a special Notes user with administrative rights.
  - c) Exit the Lotus Notes application.
  - d) Locate the Lotus Notes **Program directory** on the server, and find the file **Notes.ini**. The default location is c:\Program Files\Lotus\notes.
  - e) Copy the file Lotus **Notes.ini** to a new file, e.g. **UMRAServiceNotes.ini**.
3. Configure a password dll file.

In order to be able to exchange session passwords between the UMRA service and the Lotus Notes environment, Notes requires a special password dll file. The Lotus Notes software will access the password dll file (part of the UMRA software) to obtain a (session) password when it requires one. This is required to prevent the Lotus Notes software from trying to produce pop-up Lotus Notes login screens, which would obstruct proper operation of the service, as it cannot respond to such pop-up windows.

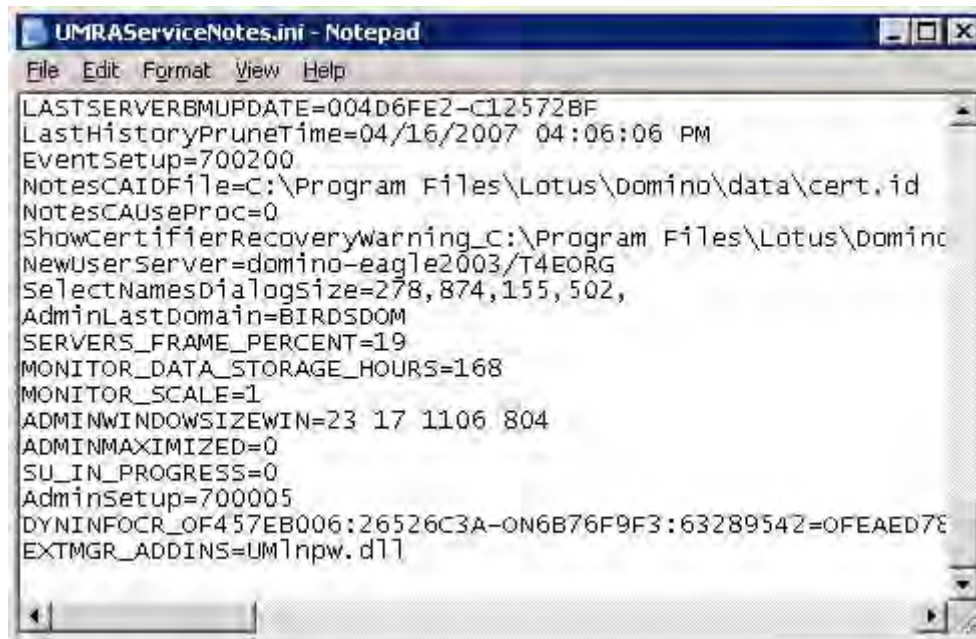
- a) Locate the **UMRA service program** directory (default c:\program files\UMRAService).
- b) This directory contains the file **UMInpw.dll**. Copy this file to the Lotus Notes program directory (i.e the directory containing the UMRAServiceNotes.ini file).
- c) Open a text editor, for instance Notepad.exe, to edit the file **UMRAServiceNotes.ini**. Navigate to the end of the file and add an extra line:

```
EXTMGR_ADDINS=UMInpw.dll
```

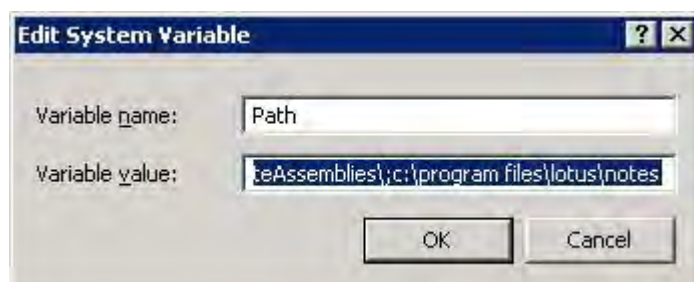
Make sure that there is still an **empty line** at the end of the file.

This configuration setting instructs the Lotus Notes software to access the specified dll (part of the UMRA software) to obtain a (session) password when it requires one.

The resulting ini file will look something like this:



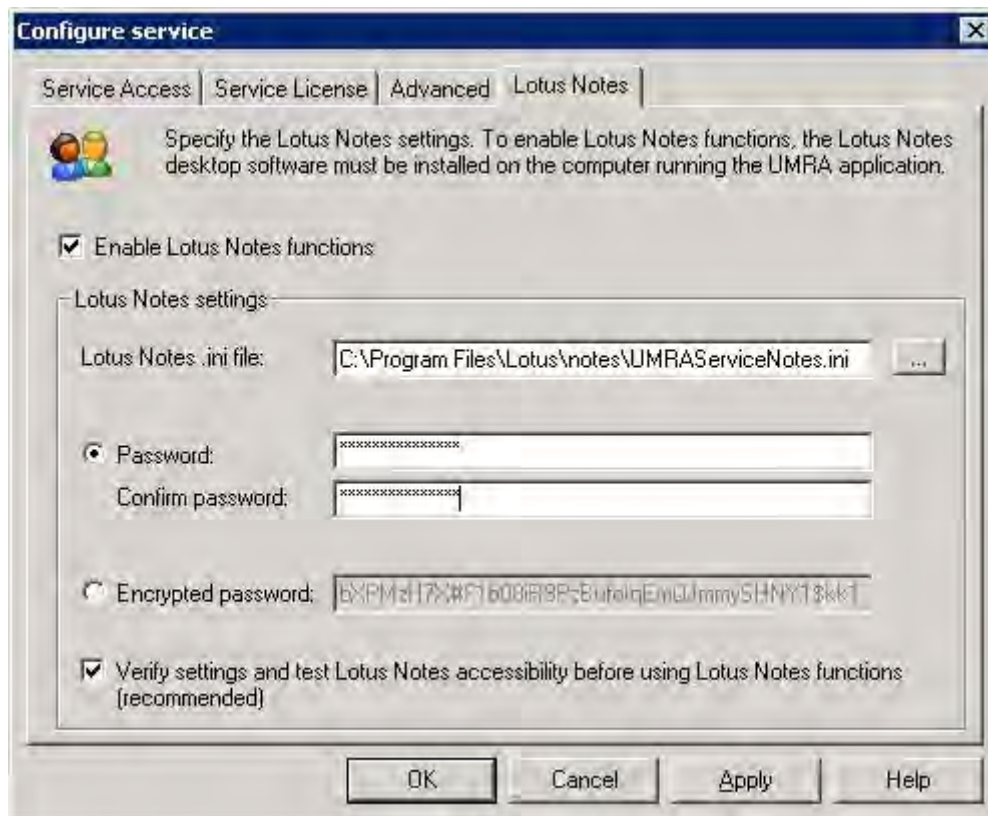
- d) Save the file, and exit the editor.
4. Add the Lotus Notes program directory to the system environment variable "Path".
  - a) Open the Windows **Control Panel**, and choose **System**.
  - b) Select the **Advanced** tab, and press the **Environment variables** button
  - c) In the bottom section, scroll to the variable **Path**, and click **Edit**.
  - d) Add here the Lotus Notes program directory (default C:\program files\lotus\notes) to the end of the "Path" variable. precede the variable with a ";" separator character.



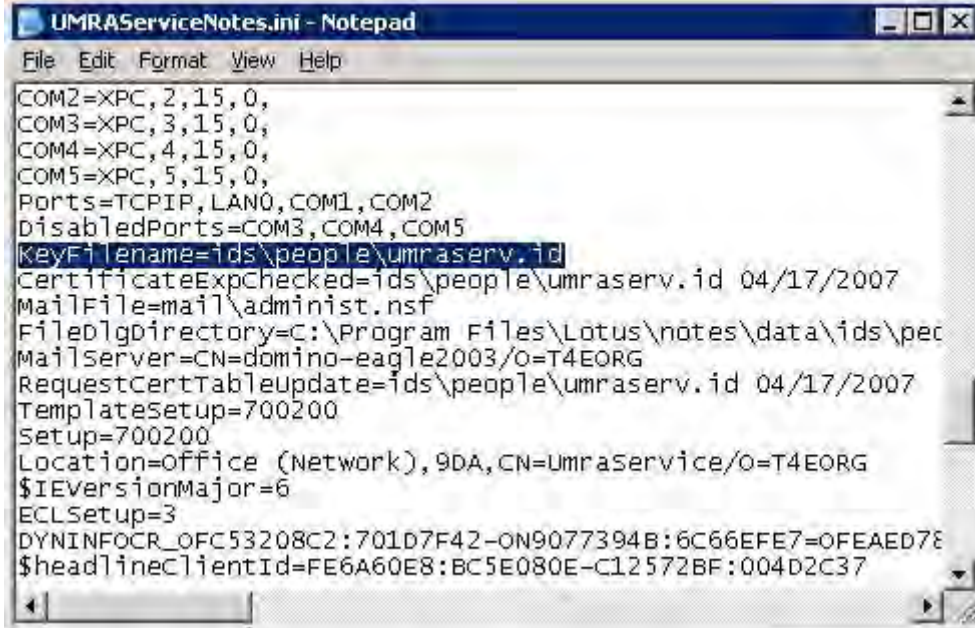
- e) **Stop and restart** the UMRA service to make sure that it will use the new settings.



5. Enable and configure the Lotus Notes configuration of the UMRA service itself by using the UMRA console.
  - a) Start UMRA console application, and select the menu **UMRA Service, connect** and connect to the UMRA service on the server you are currently configuring.
  - b) Select **UMRA Service,Service properties** and select the Lotus Notes tab



- c) Select the checkbox **Enable Lotus Notes Functions**, and specify the Lotus Notes ini file you have created in the previous steps. Specify the **password**. The password must be the password of the Lotus Notes User ID file that is listed in the **KeyFilename** property in the specified ini file. If you have followed the above procedure, this will be the Lotus Notes User Id file of the user you specified in the Lotus Notes Client.



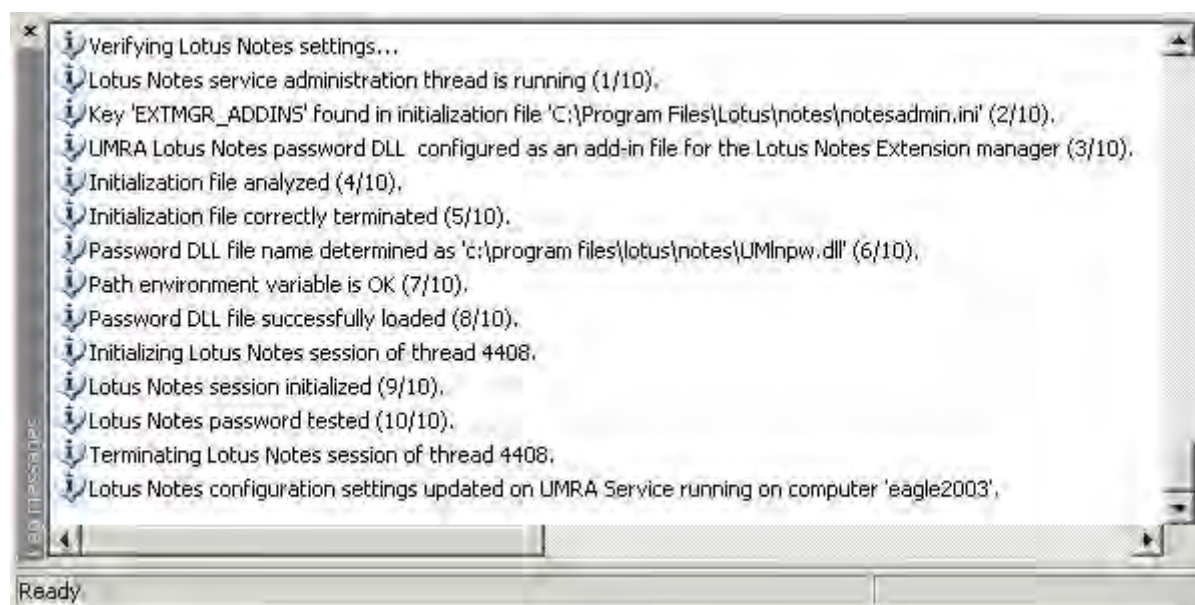
```

COM2=XPC,2,15,0,
COM3=XPC,3,15,0,
COM4=XPC,4,15,0,
COM5=XPC,5,15,0,
Ports=TCPIP,LAN0,COM1,COM2
DisabledPorts=COM3,COM4,COM5
KeyFilename=ids\people\umraserv.id
CertificateExpChecked=ids\people\umraserv.id 04/17/2007
MailFile=mail\administ.nsf
FileDlGDirectory=C:\Program Files\Lotus\notes\data\ids\pec
MailServer=CN=domino-eagle2003/o=T4EORG
RequestCertTableupdate=ids\people\umraserv.id 04/17/2007
TemplateSetup=700200
Setup=700200
Location=Office (Network),9DA,CN=UmrAServ/O=T4EORG
$IEVersionMajor=6
ECLSetup=3
DYNINFOCR_OF53208C2:701D7F42-ON9077394B:6C66EFE7=OFEAED7E
$headlineClientId=FE6A60E8:BC5E080E-C12572BF:004D2C37

```

Note that the Lotus Notes client has created many items in the .ini file some are required for the UMRA connection, many others are not. However, unless you are very familiar with their meaning, we suggest not altering this file manually. If for some reason you want UMRA to connect to the Lotus Notes environment with a different user ID, please repeat the steps 2 and 3.

- d) Press OK to continue.
- e) If you now examine the **log messages** in the reported by the service(default location C:\Program Files\UMRAServ\Log\UMRASvcLog1.txt), you should see something like:



- Now the service is successfully initialized, you can start using the Lotus Notes actions in your service based UMRA projects.

#### Configuration of the UMRA Console for use with Lotus Notes

If you want to use the Lotus notes functionality of UMRA also when using local (mass) projects executed by the console the UMRA, you also need to configure the UMRA console for use with Lotus Notes. The method of configuring the UMRA Console for use with Lotus Notes requires comparable steps as configuring the UMRA service. See *Configuring the UMRA Console for use with Lotus Notes* on page 37 for a description how to do this.

#### 3.9.3. Administration Requests database

The **Administration Requests database** in Lotus Notes (admin4.nsf) handles specific actions from the Lotus Domino Administrator. Many management tasks can be accomplished using the **Administration Requests database**. When an **Administration Request** document is submitted to the **Administration Request** database, it is processed by the **Administration Process**. The contents of the **Administration Request** document defines the exact actions executed.

Not all actions available in the Lotus Domino Administrator are implemented directly in UMRA. However, the actions in the Lotus Domino Administrator that are generating a **Administration Request** document, can be created manually with UMRA. These are a few actions that are not standard actions in UMRA but can be created manually:



- *Move mail files to another server* on page 48
- *Create replica* on page 51

To create a Administration Request document, the following actions are used in an UMRA project:

1. **Get database**  
To get the admin4.nsf database. For the **Database path** property, specify **admin4.nsf**, to access the **Administration Requests** database.
2. **Create document**  
To create a new empty document in the database. For property Form name, specify AdminRequest.
3. **Set item(s)**  
Set the appropriate items in the new document.
4. **Sign/Unsign document**  
Finally, to sign the new document.

#### Move mail files to another server

This topic describes the fields of the Administration Request document to move the mailbox of a person to another Lotus Notes Domino server.

Item name	Item type	Options	Item creation flags	Value	Example
FullName	text	Append if exist	sign names authors auto-summary	The name of the person that issues the request.	CN=Administrator/O=tools4ever
ProxyAction	text	Append if exist	sign protected auto-summary	The value specifies the action to create a replica and must equal 45.	45

ProxyAuthor	text	Append if exist	sign protected auto-summary	The name of the person that issues the request.	CN=Administrator/O=tools4ever
ProxyDatabasePath	text	Append if exist	sign protected auto-summary	The path of the mail file you want to move. The path must be relative to the Domino installation directory	mail\jsmith
ProxyDestinationDatabasePath	text	Append if exist	sign protected auto-summary	The path where you want to move the mail file to. The path must be relative to the Domino installation directory	mail\jsmith
ProxyDestinationServer	text	Append if exist	sign protected auto-summary	The name of the server you want to move the mail file to.	CN=Marketing/O=tools4ever
ProxyLinkDestinationToSCOS	text	Append if exist	sign protected auto-summary		0
ProxyNameList	text	Append if exist	sign protected auto-summary	The name list of the mail file you want to move.	CN=John Smith/O=Tools4ever

ProxyOriginatingAuthor	text	Append if exist	sign names authors auto-summary	The name of the person that issues the request.	CN=Administrator/O=tools4ever
ProxyOriginatingOrganization	text	Append if exist	sign names authors auto-summary	The name of the organization	tools4ever
ProxyOriginatingRequestUNID	text	Append if exist	auto-summary	Leave this property blank	
ProxyOriginatingTimeDate	date-time	Append if exist	sign auto-summary	The current date-time value	
ProxyOverrideDefaultDataStore	text	Append if exist	sign protected auto-summary		0
ProxyProcess	text	Append if exist	sign protected auto-summary	The name of the task that is processing the administration request.	Adminp
ProxyServer	text	Append if exist	sign protected auto-summary	The name of the server you want to send the request to.	CN=Development/O=tools4ever
ProxySourceServer	text	Append if exist	sign protected auto-summary	The name of the server you want to send the request to.	CN=Development/O=tools4ever

Type	text	Append if exist	protected auto-summary	The title of this administration process request document	AdminRequest
------	------	-----------------	------------------------	---	--------------

### Create replica

This topic describes the fields of the Administration Request document to create a replica of a Lotus Notes database.

Item name	Item type	Options	Item creation flags	Value	Example
FullName	text	Append if exist	sign names authors auto-summary	The name of the person that issues the request.	CN=Administrator/O=tools4ever
ProxyAction	text	Append if exist	sign protected auto-summary	The value specifies the action to create a replica and must equal 45.	32
ProxyAuthor	text	Append if exist	sign protected auto-summary	The name of the person that issues the request.	CN=Administrator/O=tools4ever
ProxyCopyAcl	text	Append if exist	sign protected summary	You can copy the Acl of the source database to the replica. Specify 1 to do so, or 0 to leave the Acl.	1

ProxyCreateFullTextIndex	text	Append if exist	sign protected summary	You can create a Full Text index search on the replica of the database. Specify 1 to do so, or 0 for no full text index.	0
ProxyDatabasePath	text	Append if exist	sign names summary	The location of the replica of the database. The path must be relative to the Domino installation directory	mail\replica\jsmith.nsf
ProxyDatabaseSourcePath	text	Append if exist	sign names summary	The path to the source database. The path must be relative to the Domino installation directory.	mail\jsmith.nsf
ProxyDestinationServer	text	Append if exist	sign names summary	The name of the server you want to create the replica databases on.	CN=Marketing/O=tools4ever
ProxyNameList	text	Append if exist	sign names summary	The name list of database you want to create a replica of.	John Smith
ProxyOriginatingAuthor	text	Append if exist	sign authors summary	The name of the person that issues the request.	CN=Administrator/O=tools4ever

ProxyOriginatingOrganization	text	Append if exist	sign authors summary	The name of the organization	tools4ever
ProxyOriginatingRequestUNID	text	Append if exist	auto-summary	Leave this property blank	
ProxyOriginatingTimeDate	date-time	Append if exist	sign auto-summary	The current date-time value	
ProxyOverrideDefaultDataStore	text	Append if exist	sign protected summary		0
ProxyProcess	text	Append if exist	sign protected auto-summary	The name of the task that is processing the administration request.	Adminp
ProxyReplicaId	date-time	Append if exist	sign names summary	The current date-time value	
ProxyServer	text	Append if exist	sign names summary	The name of the server you want to send the request to.	CN=Development/O=tools4ever
ProxySourceServer	text	Append if exist	sign names summary	The name of the server you want to send the request to.	CN=Development/O=tools4ever
Type	text	Append if exist	protected auto-summary	The type of this administration process request document	AdminRequest

### 3.9.4. Lotus Notes example projects

UMRA is shipped with several example projects that focus on Lotus Notes functions. The example projects can be found in subdirectory

**.\Example Projects\LotusNotes**

of the UMRA Console application. When installed on C:\ with all settings equal to their default values, the location is:

**C:\Program Files (x86)\Tools4ever\User Management Resource Administrator\Example Projects\LotusNotes**

The following Lotus Notes projects are available:

#### **Create database replica**

The project script creates and **Administration Request** to make a replica of a Lotus Notes database. The Administration Request is described in topic *Create replica* on page 51.

#### **Move to another server**

The project script creates and Administration Request to move the mail database of an account to another Lotus Notes Domino server. The administration request is described in topic *Move mailfiles to another server* on page 48.

#### **Request move to new certifier**

Illustrates the usage of action **Move person (advanced)**. The project script moves a person to another certifier.

#### **Update profile document**

Illustrates the usage of action **Update profile document**. The project script sets the Owner field of the CalendarProfile document of a mailbox database to a specified person.

#### **Approve Mail file Deletion**

Shows how to automate the confirmation of the deletion of a mail file. The example project deletes a person from the Lotus Notes Domino directory, including the mail file of the user. As a response, an approval request to confirm the mail file deletion is generated by the Lotus Notes administration process. Once the UMRA project has deleted the user from Lotus Notes, the project starts looking for the approval request in the administration process database. This may take up to a few minutes. When the approval request is found, the action *Execute agent script* on page 485 is used to approve the request. (example project file: `.\Example Projects\LotusNotes\LotusNotesApproveMailfileDeletion.xml`)

### **Remove Roaming Profile**

Shows how to setup a administration request to remove the roaming profile of a user account. If the roaming profile files are replicated to other Domino servers, these files will be deleted as well. When the administration request is processed, the administration process will create approval requests. To complete the request, these requests need to be approved. (example project file: `.\Example Projects\LotusNotes\LotusNotesRemoveRoamingProfile.xml`)

### **Lotus Notes ID Vault - Reset password**

Starting with Lotus Notes Domino 8.5, it is possible to reset the password of Lotus Notes user account using the so called ID Vault. The example project *LotusNotesVault* shows how to use UMRA to reset password using the ID Vault.

### **ID Vault**

The Lotus Notes ID Vault is available since Lotus Notes Domino 8.5. With the ID Vault it is possible to reset the password of a user account in Lotus Notes. When the password is reset, the ID of the account is updated in the ID Vault and is downloaded the next time the user logs on to Lotus Notes. With UMRA, it is possible to use the ID Vault to reset the passwords of Lotus Notes user accounts. UMRA uses a project that creates 2 Lotus Notes agents to accomplish this. This topic describes the example project that uses the ID Vault.

### **Example project**

The example project is stored in file `LotusNotesVault.xml` in the Lotus Notes example project directory. To focus on the UMRA interaction with the Lotus Notes Vault, the project is implemented as a scheduled



project for a specific user. In practice, the user account will not be fixed and can be obtained from a form, database, file, etc.

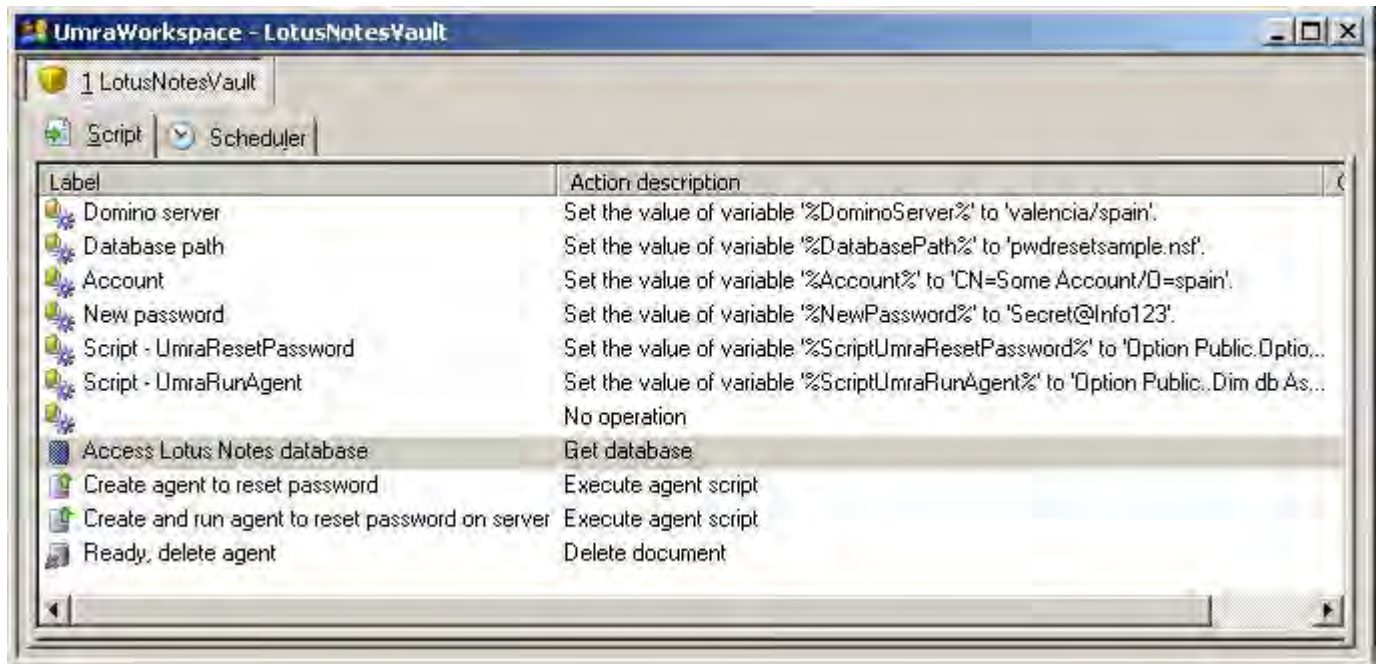
## Requirements

The following requirements apply in order to implement this or a similar project successfully:

1. The ID Vault must be operational and created with Domino 8.5 or higher. See IBM's documentation on Domino server for more information;
2. The ID's of the user accounts for which it should be possible to reset the passwords must be stored in the ID Vault. Normally, a policy is used to store the ID's in the vault;
3. The Lotus Notes used by the UMRA software to access Lotus Notes should be authorized to reset the passwords. Since the software resets the passwords using an agent, the account must be a so called **Self-service password reset authority**. Note also that the server on which the ID Vault resides must have access as a **Self-service password reset authority**.

## Project description

The project uses Lotus Notes sample database pwdresetsample.nsf. In this database, 2 Lotus Notes agents are created. The first agent (UmraResetPassword) resets the password of an account. Since this agent must be executed on the server the second agent is used. (By default, a Lotus Notes agent is executed in the Lotus Notes session environment). The second agent executes the first agent on a specific server.



The Lotus script code of the agents is straightforward. The first agent creates a session and resets the password of the account specified by a variable:

Lotus script code of first agent, UmraResetPassword:

```
Option Public
Option Declare

Sub Initialize
Dim Session As New NotesSession
  Call
Session.ResetUserPassword( "%DominoServer%", "%Account%", "%NewPassword%"
)
End Sub
```

In UMRA, the action to manage this agent only creates the agent. The agent is not executed and not deleted. The NoteID of the note holding the agent is returned to be able to delete the agent later on. The second agent accesses the first agent. Next, it executes the agent on the server.

Lotus script code of second agent, UmraRunAgent:

```
Option Public
```

```
Dim db As NotesDatabase
```

```
Dim agent As NotesAgent
```

```
Sub Initialize
```

```
    Dim s As New NotesSession
```

```
    Set db = s.CurrentDatabase
```

```
    Set agent = db.GetAgent("UmraResetPassword")
```

```
    agent.RunOnServer
```

```
End Sub
```

The second agent is deleted as part of the action that creates and runs the agent. Finally, the second agent is deleted with action **Delete document**.

### 3.10. Exchange 2007

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### 3.10.1. Introduction Exchange 2007

The main purpose of managing an Exchange 2007 environment includes the management of mailboxes, user accounts, mailbox databases, mailbox stores, Exchange servers and so on. UMRA supports all of these management tasks but focuses on the management of Exchange 2007 mailboxes and user accounts.

Starting with Exchange 2007, the Exchange environment is completely managed through Microsoft Powershell. Hence, UMRA includes the support of Microsoft Powershell and allows end-users to execute Powershell commandlets through a number of ways. For the end-user, the execution of Powershell commandlets by using UMRA is transparent: a set of normal UMRA actions are available to manage Exchange 2007 mailboxes and other Exchange 2007 resources. In the background, Powershell commandlets are executed when an UMRA script is executed to manage Exchange 2007.

To support Exchange 2007, UMRA uses a special service to allow the execution of Powershell commandlets: the Powershell Agent service. This service is part of UMRA and is required in order to manage Exchange 2007 with UMRA. The Powershell Agent service must be installed on a computer that runs both the .NET framework and the Exchange Management Console software. The computer can be a 32-bit or 64-bit machine.

### 3.10.2. Requirement UMRA Exchange 2007 support

UMRA supports a number of actions, dedicated to manage Exchange 2007 mailboxes and accounts. These actions are available for any UMRA project, including forms, automation and form projects.

In order to successfully use these actions in UMRA project scripts, the following requirements apply:

1. A separate UMRA license is required to support the UMRA actions that require Powershell support. This is the case for all UMRA Exchange 2007 actions.
2. The Powershell Agent service must be setup and configured. The configuration of UMRA to support Exchange 2007 includes the setup and configuration of the Powershell Agent service. Once the Powershell Agent service is setup, UMRA scripts with UMRA actions to manage Exchange 2007 can be used. The Powershell Agent service must be configured on a machine that has the Exchange 2007 Management software installed
3. The UMRA application that executes the UMRA project scripts, must have a connection configured with the Powershell Agent service. Note that for mass projects, the UMRA application is the UMRA Console. For all other projects, the UMRA application is the UMRA Service.

For more information on how to setup the Powershell Agent service, see *Powershell Agent service setup*. For more information on the available UMRA action to manage Exchange 2007 mailboxes, accounts and resources, see the online help topics for each action.

### 3.10.3. Manage Active Directory with the UMRA Powershell Agent service

A number of UMRA dynamic actions require the **Exchange 2007 Management Tools** to be installed, although these actions do not manage Exchange 2007 related resources. For these actions, it is not necessary to have Exchange 2007 Server installed on any server, but the Exchange 2007 Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service. Examples of these UMRA actions are: **Get AD permissions** (action folder: Powershell, Active Directory permissions) and **Get (nested) group memberships** (action folder: Powershell, Group management). These actions are implemented using cmdlets that are part of the Exchange 2007 *snap-in*

**Microsoft.Exchange.Management.PowerShell.Admin**. Therefore, the Exchange Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service in order to use these actions. Note that not all cmdlets of this snap-in can be used if Exchange 2007 server is not installed. For instance, the cmdlets to create a mail-enable user account requires Exchange 2007 Server to be installed.

The described UMRA dynamic actions have the following characteristics:

1. The actions use cmdlets that are part of the Exchange 2007 Powershell snap-in that comes with the Exchange 2007 Management Tools;
2. The cmdlets do not require Exchange 2007 Server to be installed in the network. Instead, the cmdlets can be used to manage Active Directory resources;
3. To execute the Powershell scripts that use these cmdlets, the Exchange 2007 Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service.

The Exchange 2007 Management Tools are available for 32-bit and 64-bit platforms. When Exchange 2007 Server is installed (64-bit platform only), the tools are installed automatically. To install the tools on a 32-bit platform, see *Setting up the Exchange 2007 Management Tools on a 32-bit platform* for more information. To install the tools on a 64-bit platform without installing Exchange 2007 Server, a similar procedure must be used.

### 3.10.4. Managing Exchange 2003 with the UMRA Powershell Agent service

With the UMRA Powershell Agent service, it is possible to manage a number of Exchange 2003 mailbox settings, including Exchange 2003 mailbox permissions. Special configuration settings apply to support this type of functionality.

#### Environment

This section describes the principle of the required environment to support the functions to manage Exchange 2003 with UMRA and the Powershell Agent service. The environment runs Active Directory on

Windows 2003, and one or more Exchange 2003 servers. Note that there is no server running Exchange 2007. The following systems are part of the environment:

1. **Domain controller:** There must be at least a one domain controller to run Active Directory. The domain controller must run Windows 2003 in native mode.
2. **Exchange 2003:** One or more servers run Exchange 2003 Server.
3. **UMRA:** The UMRA software (Console and Service) can be installed on any computer, except for the Powershell Agent service.
4. **UMRA Powershell Agent service:** The Powershell Agent service is installed by using the UMRA Console application. The service cannot be installed on a computer that runs Exchange 2003 server but it can be installed on any other server that is part of the domain. On this computer, the **Exchange 2007 Management Tools** must be installed. For a procedure to install the tools on a 32-bit platform, see *Setting up the Exchange 2007 Management Tools on a 32-bit platform*.

Not all cmdlets can be used to manage Exchange 2003 mailboxes. Valid cmdlets are: Get-Mailbox, Get-User, Add-MailboxPermission, Add-AdPermission, Set-Group. As a general rule, the cmdlets that access Active Directory instead of the the Exchange 2007 server can be used to manage Exchange 2003 mailboxes.

### **3.10.5. Setting up the Exchange 2007 Management Tools on a 32-bit platform**

#### **How to setup the Exchange 2007 Management Tools on a 32-bit platform**

The **Exchange 2007 Management Tools** are available both for 32-bit and 64-bit platforms. **Exchange 2007 Server** is available only for 64-bit platforms, but the **Exchange 2007 Management Tools** run on both 32-bit and 64-bit platforms. This topic describes how to setup the 32-bit **Exchange 2007 Management Tools** on a 32-bit Windows 2003, Service Pack 2 platform. Note that the **Exchange 2007 Management Tools** can also be installed on Windows XP. For this platform, the procedure is similar.

1. Log on to the computer with domain administrative access. Make sure the domain administrator is an administrator of the local computer as well.

2. If not installed, configure the **Microsoft Internet Information Services Common Files** using **Control Panel, Add or Remove Programs**. Select item **Application Server**, click **details**, select **Internet Information Services (IIS)**, click **details** and check item **Common Files**. (If the item is already checked, the **Common Files** are already installed.) Click **OK** and **Next** a number of times to confirm the selection and start the installation.

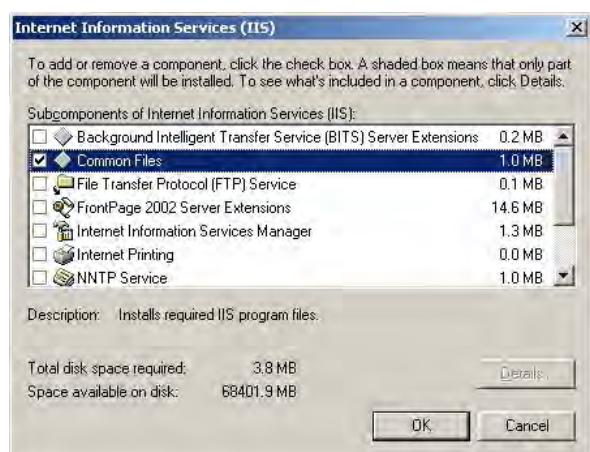


Figure: IIS common files.

3. Verify the **domain functional level**. The level must be **Windows 2000 native mode** or above. To raise the level, run **Active Directory Users and Computer** on a domain controller, right click the domain and select option **Raise Domain Functional Level**.
4. Exchange 2003 only: Verify the **operation** mode of the Exchange Organization. In the **Exchange System Manager**, right click on the organization and select menu option **Properties**. Check the contents of the **Operation Mode** field.
5. Install the pre-requirements component: **Microsoft .NET Framework Version 2.0**. To download, visit link <http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en>
6. Install the pre-requirements component: **Microsoft .NET Framework Version 2.0** hotfix. To download, visit link <http://www.microsoft.com/technet/prodtechnol/exchange/Analyzer/729d1648-ff17-43f9-a1cf-4285a82d4917.mspx?mfr=true>
7. Install the pre-requirements component: **Microsoft Management Console (MMC) 3.0**. To download, visit link <http://www.microsoft.com/downloads/details.aspx?familyid=4C84F80B-908D-4B5D-8AA8-27B962566D9F&displaylang=en>
8. Install the pre-requirements component: **Windows PowerShell**. To download, visit link <http://www.microsoft.com/downloads/details.aspx?familyid=10EE29AF-7C3A-4057-8367-C9C1DAB6E2BF&displaylang=en>
9. Download the 32-bit Microsoft Exchange 2007 installation files from: <http://www.microsoft.com/downloads/details.aspx?FamilyId=444C259E-605F-4A82-96D5-A2F448C9D4FF&displaylang=en>
10. Extract the files and start the setup.



## 11. Selection option **Install Microsoft Exchange**.

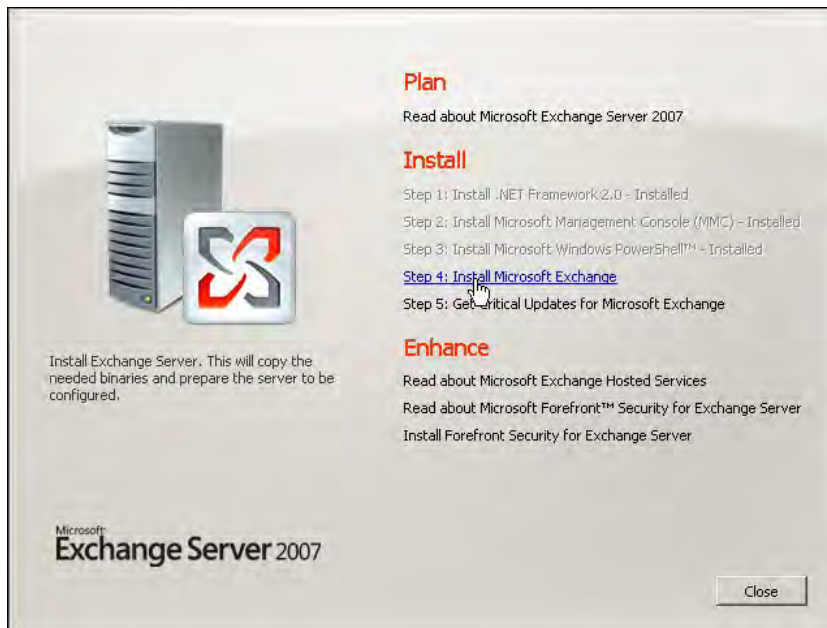


Figure: Exchange Server 2007 Init

## 12. Proceed with the wizard and when asked, selection the option **Custom Exchange Server Installation**.

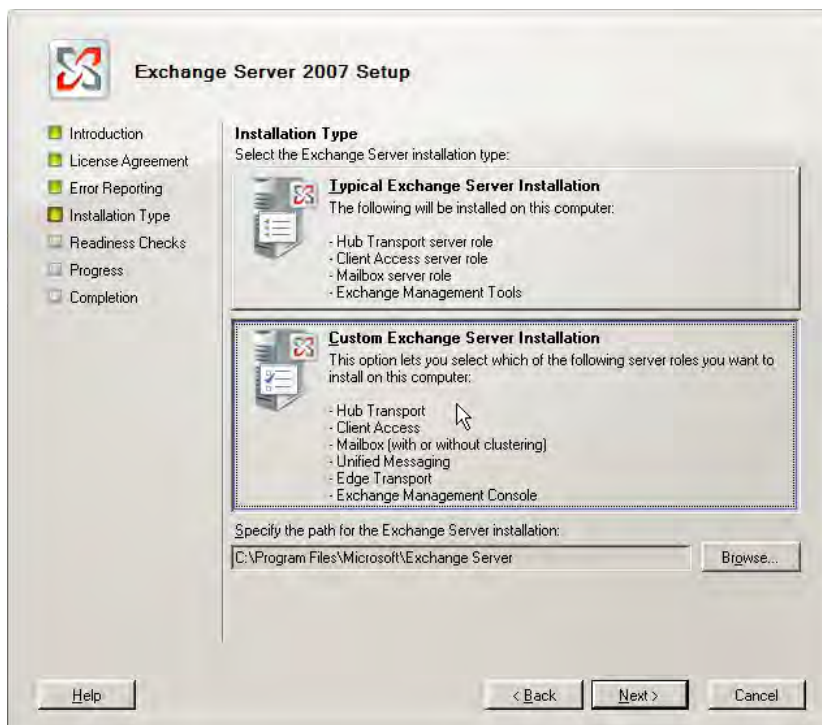


Figure: Exchange Server 2007 Installation type

13. To install only the **Management Tools**, select the appropriate option.

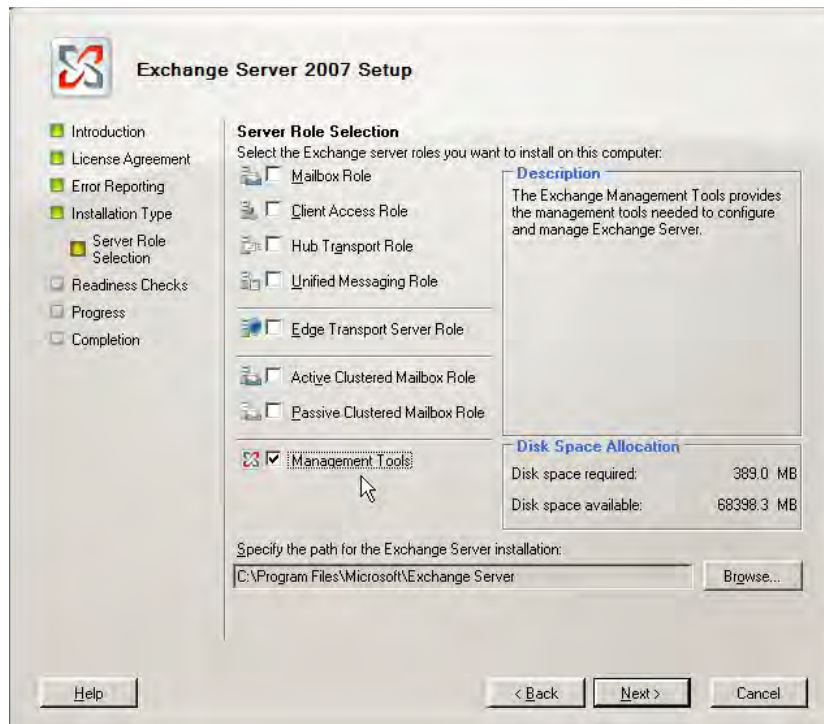


Figure: Exchange Server 2007 server role selection.

14. First, some tests are performed. When ready, select option **Install**.

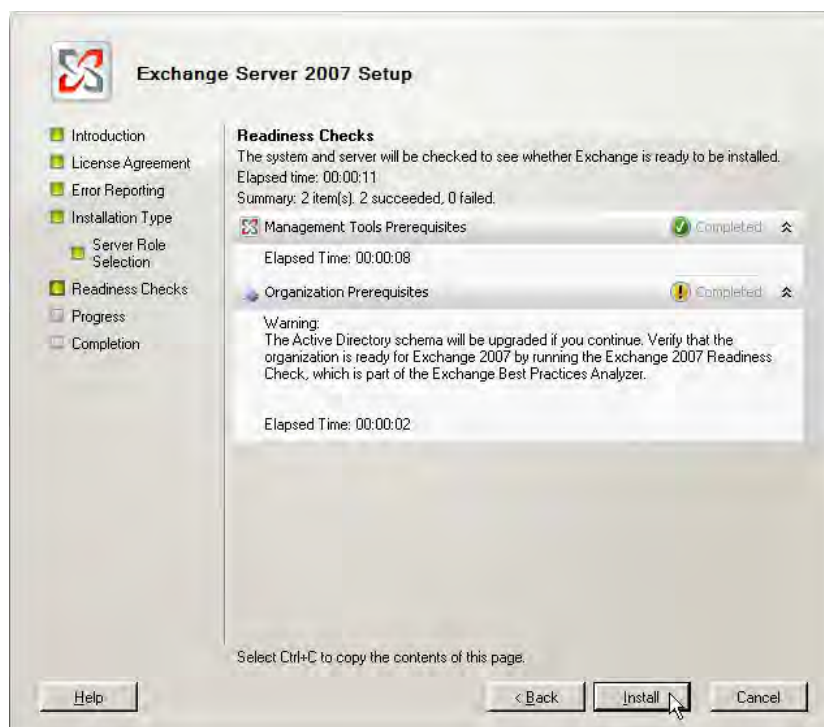


Figure: Exchange Server 2007 readiness checks

15. The **Exchange 2007 Management Tools** are now installed.

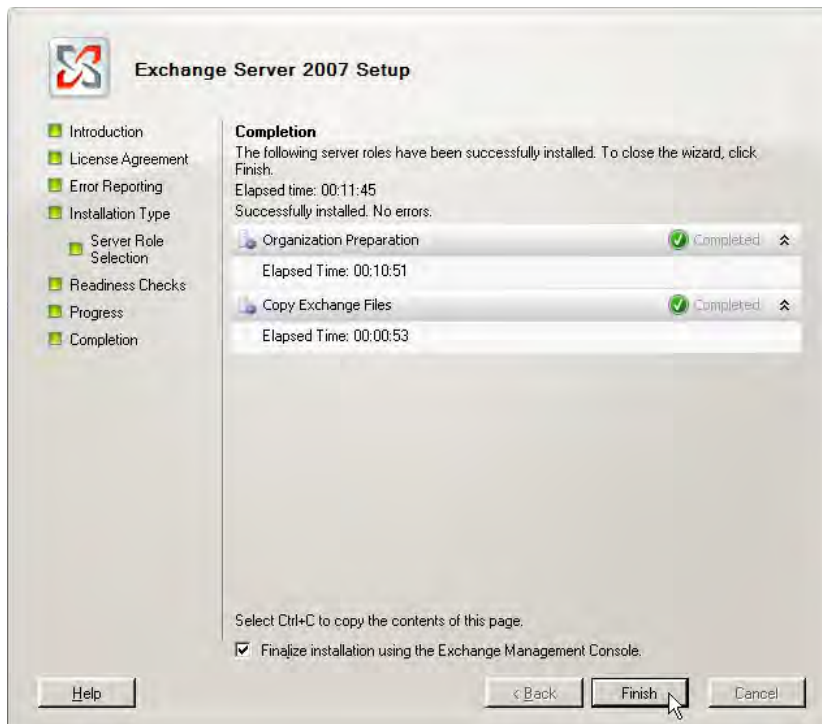


Figure: Exchange Server 2007 completion

16. When read, click **Finish** and exit the application. To test the installation, select program **All Programs, Microsoft Exchange Server 2007, Exchange Management Shell** or **Exchange Management Console**.

### 3.10.6. Using the Exchange Web Services with UMRA

With the Exchange Web Services (EWS) in an Exchange 2007 environment, UMRA is able to connect to Exchange and manage particular information on the Exchange Server. Actions in UMRA, for example 'Set Out of Office info' and 'Get Out of Office info', are using the EWS. In order for these actions to work properly, the following steps must be performed in UMRA and in the Exchange environment.

- There must be an Exchange server with the Client Access role installed. See topic : *Exchange 2007 Client Access Role* on page 8
- The UMRA Powershell Agent must be configured for use with EWS. See topic: *UMRA Powershell Agent configuration* on page 9
- A certificate must be imported in UMRA. See topic: *Exchange Web Services certificate* on page 9
- Access rights must be set on mailboxes. See topic: *Access rights of mailboxes when using Out Of Office actions* on page 12.

### Exchange 2007 Client Access Role

To use Dynamic actions with UMRA, there must be a connection with the Exchange server running the Client Access role. By default, the Client Access role is always running on one or more Exchange servers in your environment. There are two ways to get the name(s) of the Exchange server(s) running the Client Access role:

1. Open the **Exchange Management Console** on the Exchange server, navigate to **Server Configuration** and click **Client Access**. The servers mentioned are running the Client Access role.



Figure: Exchange Management Console - Client Access servers

2. Open the **Exchange Management Shell** and type: **Get-ClientAccessServer** and press enter. The Exchange servers mentioned are running the Client Access role.



Figure: Exchange Management Shell, `Get-ClientAccessServer` commandlet

If none of the Exchange servers have the Client Access role, it must be installed, otherwise the dynamic actions using the Exchange Web Service will not run. Use the Microsoft Exchange 2007 installation disk to install the Client Access role on a Exchange server. Choose the custom installation and deselect everything except the Client Access role.

### UMRA Powershell Agent configuration

In order for the dynamic actions to use the Exchange Web Services (EWS) the UMRA Powershell must apply to the following rule(s):

- Powershell Agent must run on an Exchange Server with a Exchange Powershell snapin (**Microsoft.Exchange.Management.PowerShell.Admin**). A computer with the Exchange Management Shell or Exchange Management Console use the Exchange Powershell snapin.

### Exchange Web Services certificate

When you connect to Exchange Web Services (EWS) you want to be sure you have a genuine connection to that Exchange server. Therefore the Exchange server has protected EWS with a certificate. In order to set up a connection you need to accept that certificate. Therefore UMRA needs to accept this certificate as well. Use one of the following steps to import this certificate in UMRA:

1. **Get the certificate from the Exchange server and save it on a storage device**



On the Exchange server running the Client Access rule, go to **Start, Run** and type **inetmgr**. Brows to **Web sites** and search for the web site with the EWS folder.

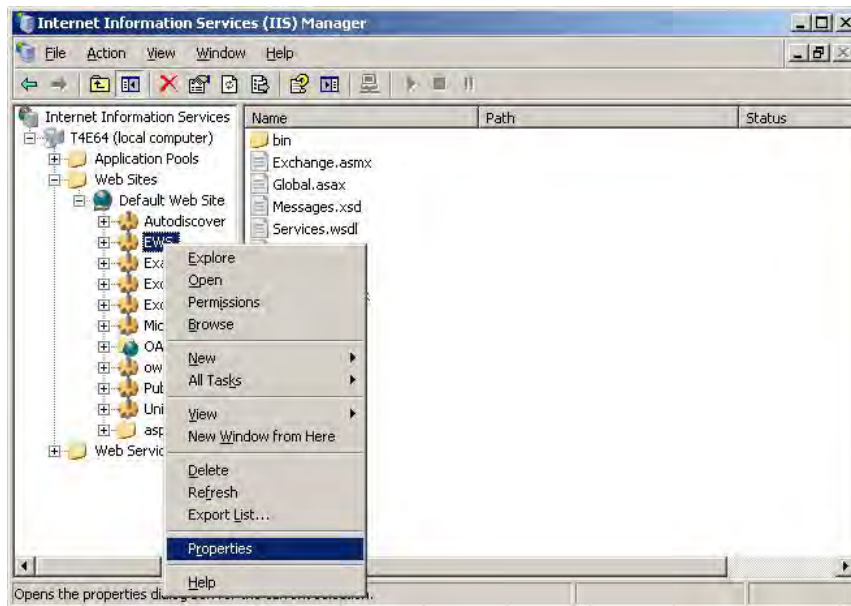


Figure: IIS - Properties of the EWS folder.

Right click on the **EWS** folder and choose **Properties**. The following dialog will appear:

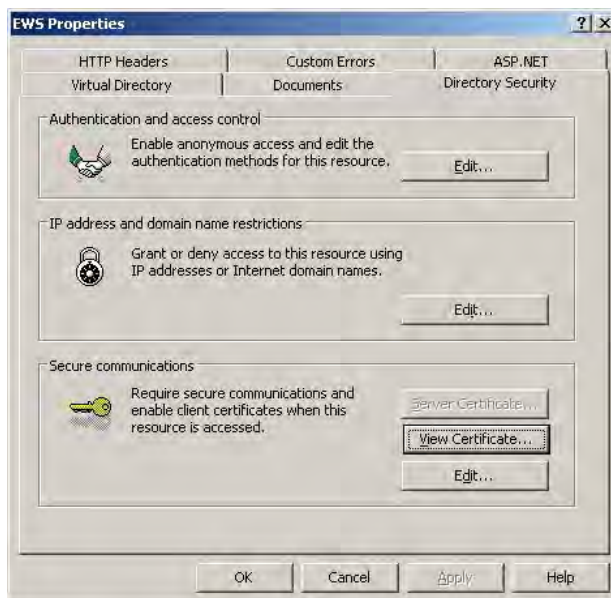


Figure: EWS Properties - Directory Security tab

On the dialog, navigate to the **Directory Security** tab and choose **View Certificate....**

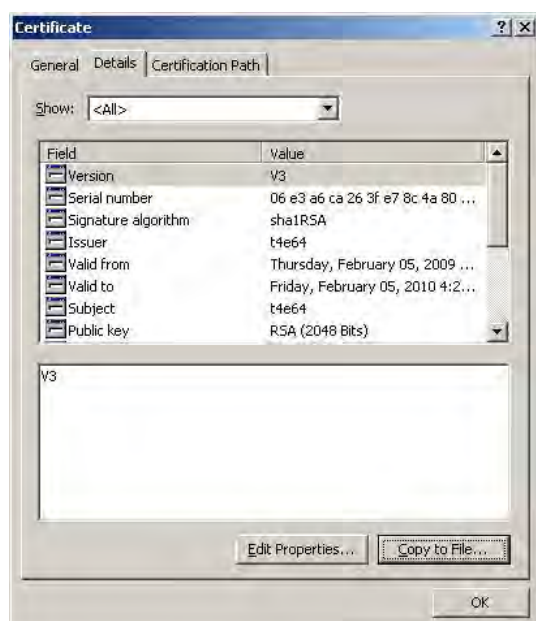


Figure: Certificate - Copy to File...

On the **Certificate** dialog, click the **Details** tab and choose **Copy to File....** Store the file as a DER encoded binary (.CER) file on your storage device and use it on the computer running UMRA.

2. **Download the certificate from the Exchange Server, for example with Internet Explorer:**

In Internet Explorer; Navigate in the menu to Tools - Internet Options - Content - Certificates. Look for a certificate with the name of your exchange server running EWS. To find Exchange servers running EWS go to: *Exchange 2007 Client Access Role* on page 8 . Click that certificate and press the button 'Export...' At the file format dialog, choose 'DER encoded binary (.CER).

If the certificate is not yet installed, navigate to the following url:

"https://exchange\_server/EWS/Services.wsdl". Replace exchange\_server by the dns name of the Exchange server running EWS. For example: netherlands.tools4ever.com. See *Exchange 2007 Client Access Role* on page 8 to find the right Exchange server.

When you've typed in the URL, Internet explorer asks whether you want to "Continue to this website (not recommended)" Click that link. Click 'View certificates' and choose to export the certificate to a DER encoded binary (.CER) file.

**Finally, import the downloaded certificate file with UMRA:**

Navigate to the Powershell Agent settings on the UMRA console (in the Menu to **Tools - Options - Powershell Agent**) or, if you use the Powershell agent with the UMRA Service, on the UMRA Service settings (in the Menu to **UMRA Service - Service properties... - Powershell Agent**)

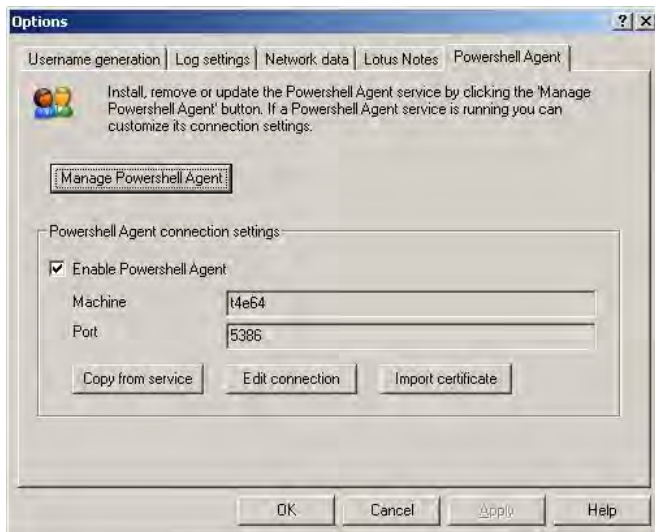


Figure: Powershell Agent settings - Import Certificate

In this dialog, click the **Import certificate** button. Now browse to the location where you saved the .CER file and click **OK**. The UMRA Log window should display an entry starting with "Certificate successfully imported..."

### Access rights of mailboxes when using Out Of Office actions

In order for UMRA to get and set Out Of Office settings of a mailbox, it will need permissions to access that mailbox. In other words, the UmraPsSvcAccount user account needs access because Exchange 2007 with UMRA is managed with Powershell. But permissions of a mailbox are quite complex. The next chapters will explain the permissions of a mailbox and how to grant read and write (get and set Out of Office) access to the UmraPsSvcAccount user account.

Note: Usually the UmraPsSvcAccount is a member of 'Domain Admins', make sure this is the case in your environment.

The error 'User is not mailbox owner' is because the UmraPsSvcAccount has not enough access and can have three causes:



1. Mailbox has never been used before and/or mail has never been sent to this mailbox. For more information about this topic go to *Mailbox has never been used* on page 13.
2. The Exchange organization object forces a deny access for Domain Admins for all mailboxes in the organization. For more information, read chapter *Deny 'Receive As' access for 'Domain Admins' on mailboxes* on page 16.
3. Mailbox is in use but access rights are not propagated to Active Directory. SOAP uses attributes in Active Directory to check the rights. For more information about this topic go to *Access rights are not propagated to Active Directory* on page 14.

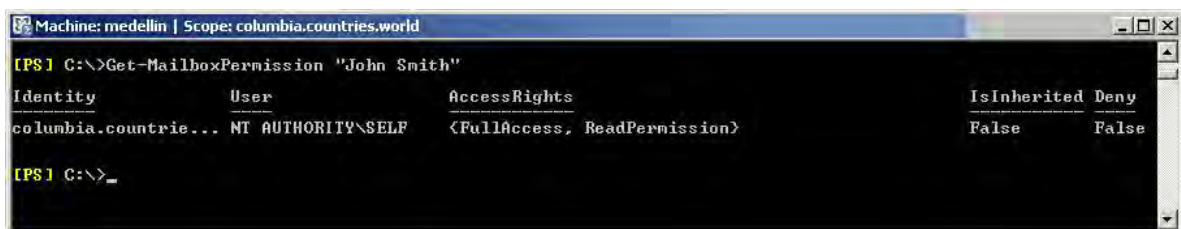
To learn more about the access rights on mailboxes, read *Background information about access rights on mailboxes* on page 21.

#### *Mailbox has never been used.*

A mailbox can be in two different states. Before it has ever been used and after it has been used. Below the explanation of the two stages. In order for UMRA to use Out Of Office, the mailbox must be in the second state.

#### 1. **The mailbox has just been created:**

When a mailbox has just been created, like the mailbox of **John Smith** in this example, it has initially only granted access to '**NT AUTHORITY\SELF**'. You can say the mailbox entry has been created but the mailbox itself does not yet exist. Because only '**NT AUTHORITY\SELF**' has access, the Powershell Agent has no access. Check the permissions by opening the Exchange Management Shell and type: `Get-MailboxPermissions "John Smith"`. When a mailbox user never has logged on or there has never been sent a mail to this mailbox, the permissions look like this:



```
Machine: medellin | Scope: columbia.countries.world
[PS] C:\>Get-MailboxPermission "John Smith"
Identity      User              AccessRights      IsInherited Deny
-----
columbia.countrie... NT AUTHORITY\SELF <FullAccess, ReadPermission> False      False
[PS] C:\>_
```

Figure: `Get-MailboxPermissions`, mailbox user has never logged on and has no mail.

#### 2. **Mailbox user has logged on or mail has been sent:**

When the mailbox user receives mail or logs on and configures Outlook, the mailbox itself will be created and all the rights of the organization in Active Directory that apply for its descendents will now apply for the mailbox as well.

To simulate that a mailbox is being used, run the Get-MailboxFolderStatistics on the mailbox. It has the same effect for the permissions as sending mail or logging on.

The mailbox of John Smith has now received an e-mail and after running the Get-MailboxPermission commandlet the output will look like this:



```
Machine: medellin | Scope: columbia.countries.world
[PS] C:\>Get-MailboxPermission "John Smith"

Identity      User              AccessRights      IsInherited Deny
-----
columbia.countrie... NT AUTHORITY\SELF <FullAccess, Rea... False False
columbia.countrie... COLUMBIA\ROSE$    <ReadPermission> True  False
columbia.countrie... COLUMBIA\Exchange... <FullAccess> True  True
columbia.countrie... NT AUTHORITY\NETW... <ReadPermission> True  False
columbia.countrie... COLUMBIA\Exchange... <FullAccess> True  False
columbia.countrie... COLUMBIA\Exchange... <ReadPermission> True  False
columbia.countrie... COLUMBIA\Administ... <FullAccess, Del... True  False
columbia.countrie... COLUMBIA\Exchange... <ReadPermission> True  False
columbia.countrie... COLUMBIA\Exchange... <FullAccess, Del... True  False
columbia.countrie... COLUMBIA\Exchange... <ReadPermission> True  False
columbia.countrie... COLUMBIA\Enterpri... <FullAccess, Del... True  False
columbia.countrie... COLUMBIA\Domain A... <FullAccess, Del... True  False

[PS] C:\>
```

Figure: Get-MailboxPermission, mailbox user has not configured Outlook or received mail.

So when the user has properly configured Outlook or someone has sent mail to that specific mailbox, the rights of the UmraPsSvcAccount can be evaluated. If, by now, rights are set properly for the mailbox and the UmraPsSvcAccount still can not get or set the Out of Office settings, go to chapter *Access rights are not propagated to Active Directory* on page 14.

For more information about the states of mailboxes and its access rights, read *Background information about access rights on mailboxes* on page 21.

#### *Access rights are not propagated to Active Directory*

The SOAP connection to the Exchange server to use the **Exchange Web Services** checks the rights of the mailbox in Active Directory. For more information about the access rights of mailboxes read topic *Background information about access rights on mailboxes* on page 21.

The inherited rights of the mailbox on Exchange are not always properly propagated to Active Directory. To force propagation, "touch" the permissions of the mailbox in Exchange. First, to check if the Active Directory security settings of the mailbox has not enough rights type:

**\$Mailbox = Get-Mailbox "John Smith" [press enter]**

`$Mailbox.ExchangeSecurityDescriptor` [press enter]. The output looks like this:



```
Machine: medellin | Scope: columbia.countries.world
[PS] C:\>$Mailbox = Get-Mailbox jsmith
[PS] C:\>$Mailbox.ExchangeSecurityDescriptor

ControlFlags      : DiscretionaryAclPresent, SelfRelative
Owner             : S-1-5-10
Group             : S-1-5-10
SystemAcl         :
DiscretionaryAcl  : <System.Security.AccessControl.CommonAce>
ResourceManagerControl : 0
BinaryLength      : 72

[PS] C:\>_
```

Figure: Display the Exchange Security Descriptor of a mailbox

To display a proper view of the name of the DiscretionaryAcl type the following:

`$Mailbox.ExchangeSecurityDescriptor.DiscretionaryAcl | Select-Object @{"Name="SecurityIdentifier"; Expression={ $_.SecurityIdentifier.Translate([System.Security.Principal.NTAccount]).value} }, IsInherited`



```
Machine: medellin | Scope: columbia.countries.world
[PS] C:\E2k7Setup>$Mailbox.ExchangeSecurityDescriptor.DiscretionaryAcl | Select-Object @{"Name="SecurityIdentifier"; Expression={ $_.SecurityIdentifier.Translate([System.Security.Principal.NTAccount]).value} }, IsInherited

SecurityIdentifier      IsInherited
-----
NT AUTHORITY\SELF      False

[PS] C:\E2k7Setup>
```

Figure: List mailbox permissions of Active Directory's Exchange Security Descriptor

Now you can see there are no inherited rights yet on the Exchange Security descriptor. After "touching" the permissions of the mailbox the rights will be propagated to the Active Directory Exchange Security descriptor as well. The safest way to do that is to add a permission that is already there. That way the permissions are still "edited" or "touched" but no changes are made. By this action the inherited properties will be propagated to the Active Directory Exchange Security descriptor of the mailbox. After "touching" the Powershell Agent user account has rights to view and edit the mailbox settings.

To "touch" the mailbox permissions for "John Smith" type:

**Add-MailboxPermission "John Smith" -User "NT AUTHORITY\SELF" -AccessRights FullAccess**



```
Machine: medellin | Scope: columbia.countries.world
[PS] C:\>Add-MailboxPermission "John Smith" -User "NT AUTHORITY\SELF" -AccessRights FullAccess
WARNING: Appropriate ACE is already present on object "CN=John Smith,CN=Users,DC=columbia,DC=countries,DC=world" for account "NT AUTHORITY\SELF".

Identity      User      AccessRights      IsInherited Deny
-----
columbia.countrie... NT AUTHORITY\SELF <FullAccess>      False      False

[PS] C:\>_
```

Figure: Add an existing mailbox permission to an account

Note: To touch the permissions for all users within a specific OU type the following and replace OU with the name of the OU you want to use;

```
Get-Mailbox -OrganizationalUnit "OU" | foreach {Add-mailboxpermission $_.DistinguishedName -
user "nt authority\self" -accessrights fullaccess}
```

Now check the permissions again by typing:

```
$Mailbox = Get-Mailbox "John Smith" [press enter]
```

```
$Mailbox.ExchangeSecurityDescriptor.DiscretionaryAcl | Select-Object @{"Name="SecurityIdentifier";
Expression={ $_.SecurityIdentifier.Translate( [System.Security.Principal.NTAccount]).value}},
IsInherited
```

The following results should appear:



Figure: List mailbox permissions of Active Directory's Exchange Security Descriptor

*Deny 'Receive As' access for 'Domain Admins' on mailboxes*

Now that the mailbox fully exists and rights on the mailbox are properly inherited, the **UmraPsSvcAccount** may still not have enough rights. In the end, the **UmraPsSvcAgent** (or **Domain Admins**, or any other group which **UmraPsSvcAccount** is a member of, depending on the policies your company maintains) needs an **'Allow' 'Receive As'** right on the mailbox. Because, by default (check if this applies to your company), the inherited rights of every created mailbox contains a **'Deny' 'Receive As'** right for **Domain Admins**. For more information, read topic Background information about permissions on mailboxes

There are two ways to give the **UmraPsSvcAccount** access to mailboxes.

1. Allow access on whole organization, for more information click: Allow access on the whole organization on page 17.
2. Allow access per mailbox user, for more information click: Allow access per mailbox on page 19

Allow access on the whole organization is the fastest and easiest action to have the **UmraPsSvcAgent** manage Out of Office. Obviously, by editing rights of the whole organization, you may issue some security problems. The second option, Allow access per mailbox user, is much more secure cause only

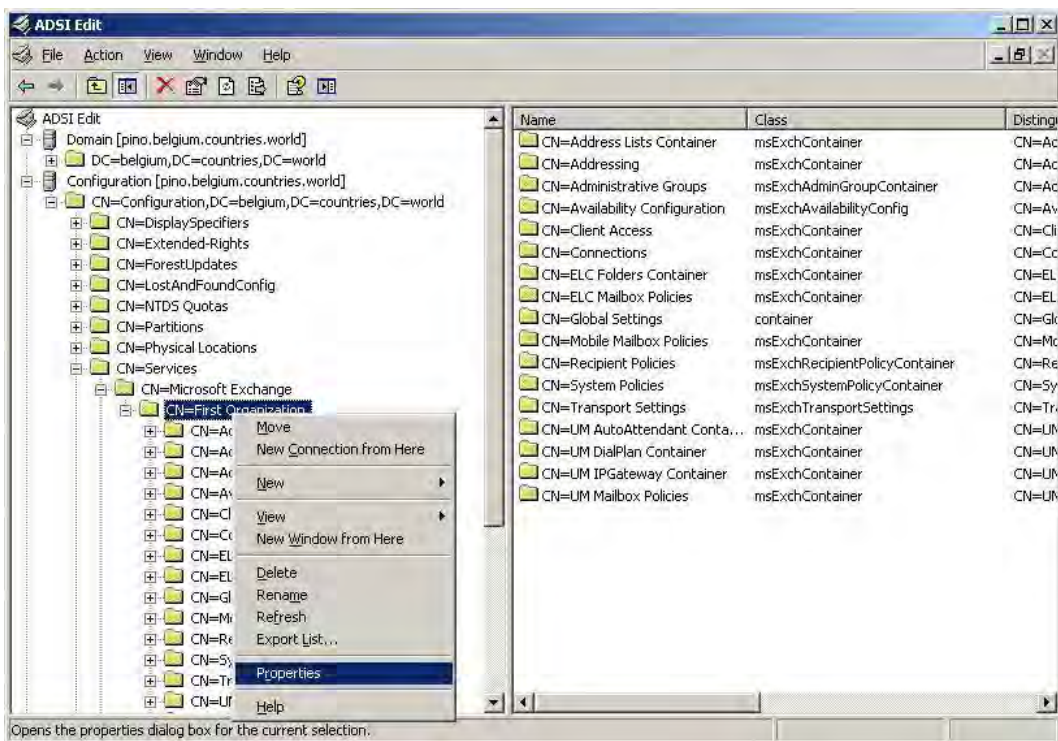
access to the specified mailbox will be given to the **UmraPsSvcAccount**. Of course you have to set the rights on each and every mailbox you want to manage, which is much more work. Read the chapters about the two options to choose which one suits your organization best.

Allow access on the whole organization

To prevent mailboxes to inherit the 'Deny' 'Receive As' right for 'Domain Admins' you must navigate to the parent object that originally has set this right. Use the application **AdsiEdit.msc** from the **Windows Support Tools** to navigate to this object, or get the object in Powershell by using the commandlet `Get-StorageGroup`, get the distinguishedName of First organization.

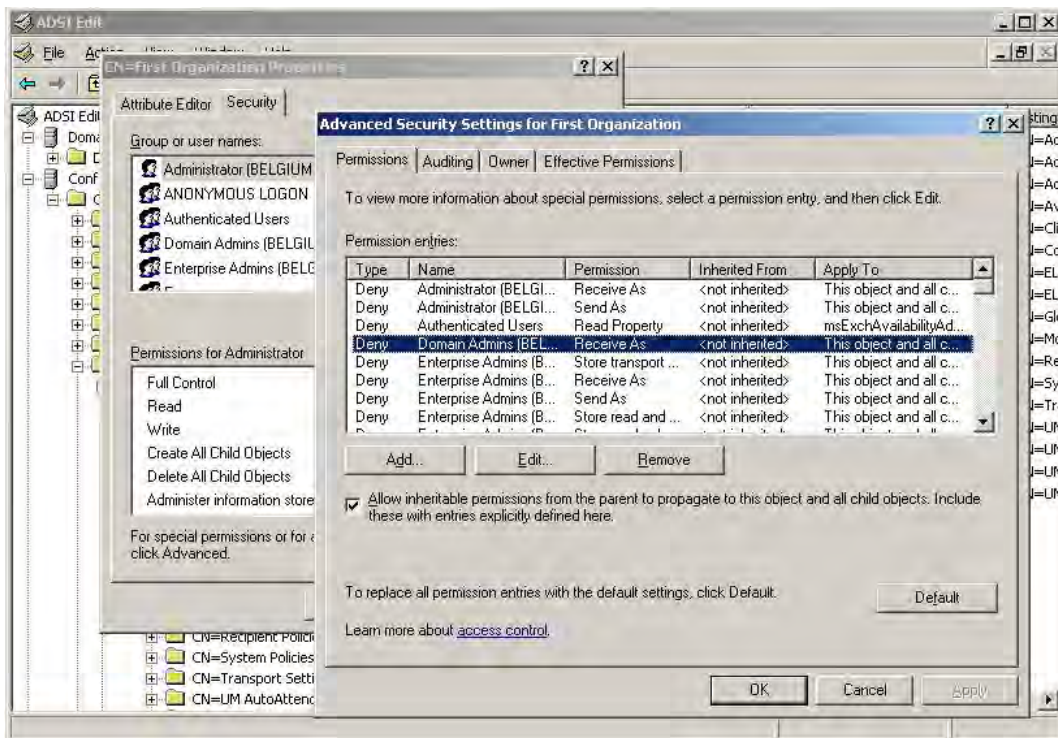
*Adsiedit:*

Navigate to Configuration + 'Your Domain' - CN=Services -CN=Microsoft Exchange – CN=First Organization. Right click the Organization object, usually called 'First Organization'.





Choose the tab 'Security' and click the Advanced button. Now locate the right with settings: **Deny=True, User=Domain Admins, Right=Receive As**. Click that record and press the remove button:



Powershell:

Get the distinguishedName of the First Organization by using the following powershell commandlet:

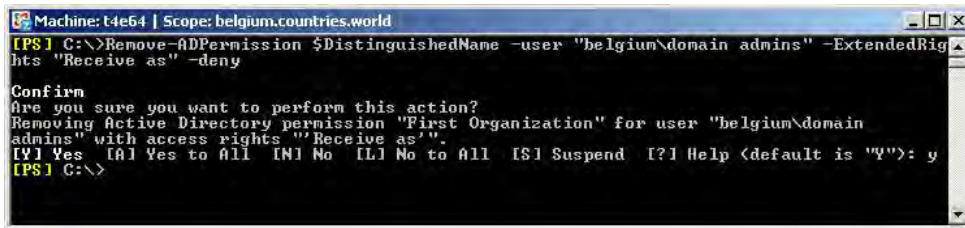
**\$DistinguishedName = (Get-OrganizationConfig).distinguishedName**

With the distinguishedName of your organization run the commandlet Get-ADPermissions



Figure: Get the AD permissions of the First Organization object.

Notice the presence of the 'Deny' 'Receive As' for 'Domain Admins'. To remove this right, use the Remove-ADPermission commandlet.



```
Machine: t4e64 | Scope: belgium.countries.world
[PS] C:\>Remove-ADPermission $DistinguishedName -user "belgium\domain admins" -ExtendedRights "Receive as" -deny
Confirm
Are you sure you want to perform this action?
Removing Active Directory permission "First Organization" for user "belgium\domain admins" with access rights "Receive as".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
[PS] C:\>
```

Figure - Remove the Deny permission on First Organization

Note: All objects that have the 'First Organization' object as a parent do not inherit that right anymore. Be aware of the consequences this can have in your organization

Allow access per mailbox user

To leave the organizations security intact and still give the **UmraPsSvcAgent** enough permissions, you must explicitly set access for the **UmraPsSvcAgent** on each mailbox itself. Explicit rights override inherited rights. To give the **UmraPsSvcAccount** rights on a mailbox, use the **Exchange Management Console** or the **Exchange Management Shell**.

*Exchange Management Console*

Navigate to the folder recipients. Right-click on the mailbox of which you want to add the permission to. Choose **Manage Full Access Permissions...** Select '**Domain Admins**', '**UmraPsSvcAccount**' or every other group the **UmraPsSvcAccount** is a member of and choose OK.

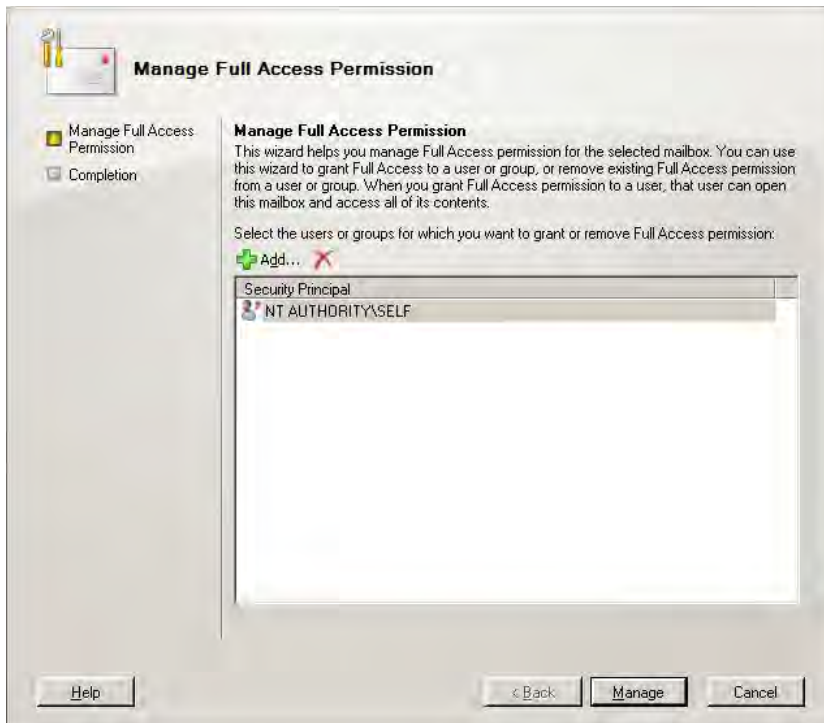


Figure: Add permissions to a mailbox with the Exchange Management Console

### Exchange Management Shell

Use the **Get-MailboxPermissions** commandlet to see whether a mailbox already gives permission to the **UmraPsSvcAccount** account. If this is not the case, use the **Add-MailboxPermission** commandlet to add the new permission to the mailbox. To give the **UmraPsSvcAccount** permission to all mailboxes in an OU that not already have the permission set, in this example the **InternalAccounts** OU, run the following script:

```
foreach ($i in Get-Mailbox -OrganizationalUnit InternalAccounts)
{
    $Permission = Get-MailboxPermission $i | Where {$_.User -like "*UmraPsSvcAccount*"}

    if($permission -eq $null)
    {
        Add-MailboxPermission $i -User UmraPsSvcAccount -AccessRight ReadPermission
    }
}
```



```

    }else

    {

        Write-Output "Mailbox of $i already has granted permission to UmraPsSvcAccount."

    }

}

```

Note that whenever a new mailbox is created, you have to explicitly add permission for the **UmraPsSvcAccount** on that mailbox. To avoid running Out of Office scripts that will partly fail because half of the mailboxes do not give access to UMRA, make sure you check every mailbox its permissions, before using Out Of Office.

## Background information about access rights on mailboxes

### About user accounts and mailboxes

A mailbox in Exchange consists of an user account in **Active Directory** and a **StoreMailbox** data object in the Mailbox Database in Exchange. These two objects are linked to each other by corresponding attributes, for example the properties of an user account in Active Directory that start with "msExch". The two object together are called a **Mailbox**. It is important to be aware that there are two objects in order to understand the permissions of a Mailbox.

### Creating a Mailbox

When you create a mailbox with the **Exchange Management Shell** or **Exchange Management Console**, two objects are created. One (the user account) in Active Directory and the **StoreMailbox** in the **Mailbox Database** in **Exchange**. It is important to know that after creation, the **StoreMailbox** is not yet fully provisioned. The object exists in **Exchange**, but the structure of the object is not yet available. To prove this, run the commandlets **Get-Mailbox** and **Get-MailboxStatistics** on the mailbox.



Figure: Proof mailbox does not fully exist after creation

If a mailbox is not used, the object does not have to be fully created. The creation of the **StoreMailbox** object will therefor be completed when used, so when the specific user configures Outlook with his account or when someone has sent an email to his mailbox.

### About the permissions:

At this stage the **StoreMailbox** object does not have any permissions yet. However, when you use the **Get-MailboxPermissions** commandlet, it shows one permission for user '**NT AUTHORITY\SELF**'. This permission is read from the AD user object's property **msExchMailboxSecurityDescriptor**, because the **StoreMailbox** does not fully exist yet.



Figure: The permissions of a mailbox when it has not been used yet.

The commandlet **Get-MailboxPermissions** uses the **msExchMailboxSecurityDescriptor** to list the permissions before the mailbox fully exists, and after that it uses the permissions of the **StoreMailbox**. To prove this you can empty the **msExchMailboxSecurityDescriptor** with Active Directory attribute editor (for instance UMRA). **Get-MailboxPermissions** will now generate an error because the object does not exist. To prove that after provisioning the mailbox **Get-MailboxPermissions** refers to the permissions in the **StoreMailbox**, run **Get-MailboxFolderStatistics** (therefor inherited rights are propagated to the **StoreMailbox**, and run **Add-MailboxPermission** to add FullAccess to user '**NT AUTHORITY\SELF**' this already exists, so the permissions will only be touched and not edited, but the **mxExchMailboxSecurityDescriptor** is updated anyway.

If you empty the **msExchMailboxSecurityDescriptor** with LDAP again, the **Get-MailboxPermissions** commandlet will now NOT generate an error but, as expected, list all the permissions. Now, apparently, the **Get-MailboxPermissions** commandlet looks up the information on the **StoreMailbox**!

### Mailbox is fully provisioned

All mailboxes inherit their rights from the **First Organization** object. You can search for this object and see its permissions when you have the **Microsoft Support Tools** installed and run **adsiedit.msc** or when using the commandlet **Get-OrganizationConfig**. To look at the rights on this object, use the commandlet **Get-ADPermission**.

▪ *AdsiEdit:*

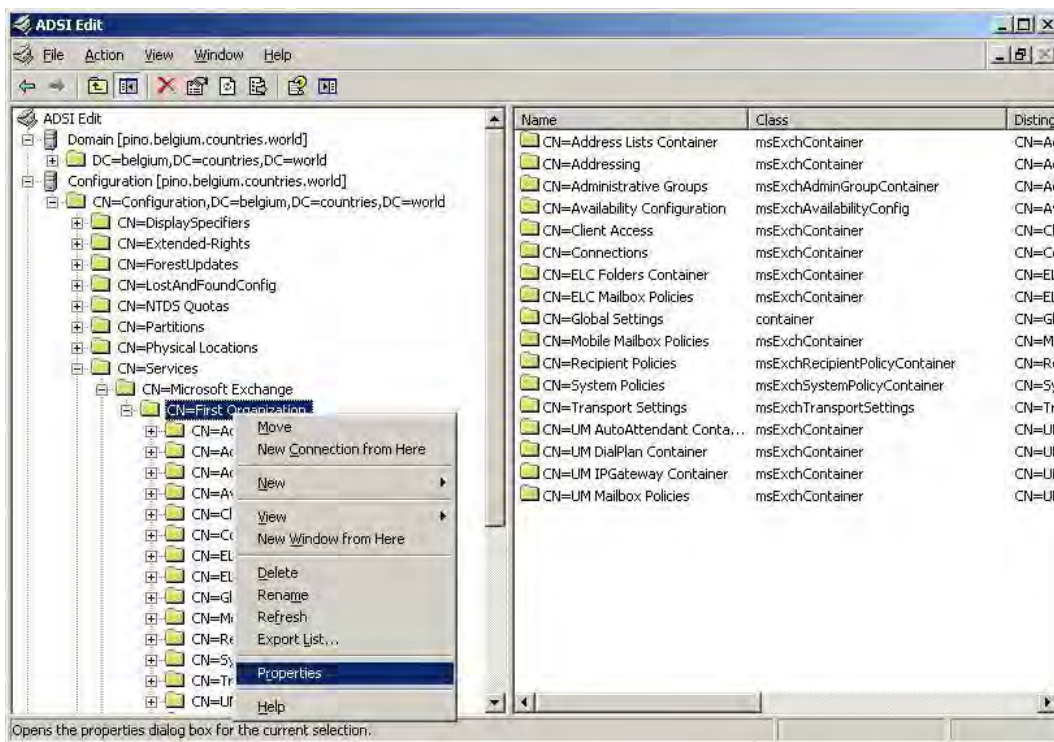
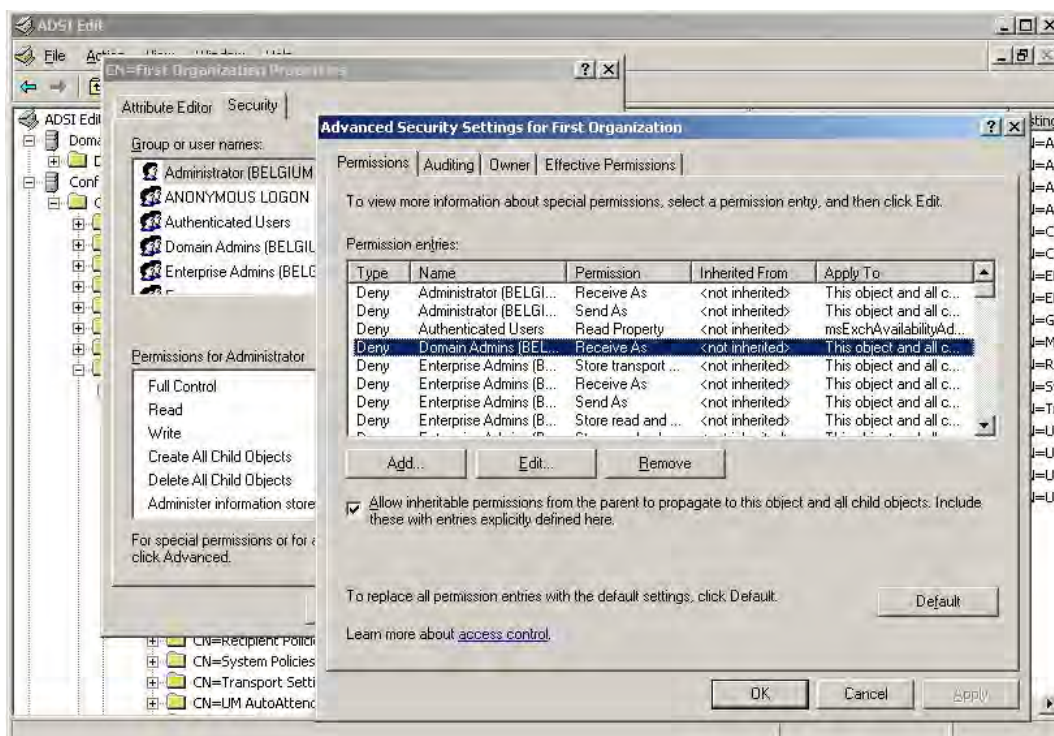


Figure: AdsiEdit and the First Organization object

To see the permissions of this object, click **Properties** and navigate to the **Permissions** tab



■ Powershell:



```
Machine: t4e64 | Scope: belgium.countries.world
[PS] C:\>$FirstOrganization = Get-OrganizationConfig
[PS] C:\>Get-ADPermission $FirstOrganization.DistinguishedName
```

Identity	User	Deny	Inherited	Rights
First Organization	NT AUTHORITY\Auth...	True	False	ReadProperty
First Organization	BELGIUM\administr...	True	False	Send-As
First Organization	BELGIUM\administr...	True	False	Receive-As
First Organization	BELGIUM\Domain Ad...	True	False	Receive-As
First Organization	BELGIUM\Schema Ad...	True	False	ms-Exch-EPI-Token-Serializati
First Organization	BELGIUM\Schema Ad...	True	False	ms-Exch-EPI-Impersonation
First Organization	BELGIUM\Enterpris...	True	False	Send-As
First Organization	BELGIUM\Enterpris...	True	False	ms-Exch-Store-Read-Write-Acce

Figure: Get a list of the permissions set on the First Organization in an Exchange 2007 environment

When a mailbox just had been created, the the StoreMailbox does not yet fully exist. This is why, at first, the only right on the mailbox is 'NT AUTHORITY\SELF'. The rights that a mailbox should inherit of First Organization are not available yet. Use the Get-MailboxPermissions commandlet to see that only one right will appear.



```
Machine: t4e64 | Scope: belgium.countries.world
[PS] C:\>Get-MailboxPermission "Tom Watson"
```

Identity	User	AccessRights
belgium.countries...	NT AUTHORITY\SELF	<FullAccess, ReadPermission>

```
[PS] C:\>
```

Figure: The permissions of a mailbox when it has not been used yet.

Now, when we use the mailbox, say, send an email to the mailbox or let the corresponding user account configure Outlook, the inherited permissions of a mailbox will appear.



Note: To simulate that a mailbox is in use, run the **Get-MailboxFolderStatistics** commandlet on the mailbox.

```
Machine: t4e64 | Scope: belgium.countries.world

[PS] C:\>Get-MailboxPermission twatson

Identity          User                AccessRights        IsInhe
-----
belgium.countries... NT AUTHORITY\SELF    <FullAccess, ReadPermission>    False
belgium.countries... BELGIUM\T4E645       <ReadPermission>                True
belgium.countries... BELGIUM\Exchange ... <FullAccess>                    True
belgium.countries... BELGIUM\administr... <FullAccess>                    True
belgium.countries... BELGIUM\Domain Ad... <FullAccess>                    True
belgium.countries... BELGIUM\Enterpris... <FullAccess>                    True
belgium.countries... BELGIUM\Exchange ... <FullAccess>                    True
belgium.countries... BELGIUM\Exchange ... <ReadPermission>                True
belgium.countries... BELGIUM\Exchange ... <FullAccess>                    True
belgium.countries... NT AUTHORITY\NETW... <ReadPermission>                True
belgium.countries... BELGIUM\administr... <FullAccess, DeleteItem, ReadPermission, ChangePermissio... True
belgium.countries... BELGIUM\Exchange ... <FullAccess, DeleteItem, ReadPermission, ChangePermissio... True
belgium.countries... BELGIUM\Exchange ... <ReadPermission>                True
belgium.countries... BELGIUM\Exchange ... <ReadPermission>                True
belgium.countries... BELGIUM\Enterpris... <FullAccess, DeleteItem, ReadPermission, ChangePermissio... True
belgium.countries... BELGIUM\Domain Ad... <FullAccess, DeleteItem, ReadPermission, ChangePermissio... True

[PS] C:\>
```

Figure: The inherited permissions of a mailbox in use

Now the **Get-MailboxPermissions** commandlet looks up its information on the **StoreMailbox** and with every update of the permissions, the **msExchMailboxSecurityDescriptor** is updated as well. But since **Get-MailboxFolderStatistics**, sending an email or logging on with Outlook is not really an update of the permissions of the **StoreMailbox**, the **msExchMailboxSecurityDescriptor** has still only the **NT AUTHORITY\SELF** right! To update the **msExchMailboxSecurityDescriptor** property, simulate adding a permission by running the **Add-MailboxPermission** commandlet and adding a permission that already exists, so no real changes are made. For example:

```
Add-MailboxPermission "Tom Watson" -User "NT AUTHORITY\SELF" -AccessRights FullAccess
```

### 3.11. Exchange 2010

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### 3.11.1. Introduction Exchange 2010

In this and the following topics you can find general information how to use Exchange 2010 actions in UMRA.

#### General

UMRA communicates (through the Tools4ever Powershell Service) by means of **Remote Powershell** with the Exchange environment. The UMRA actions are as close as possible to the individual Powershell commands available in the Exchange 2010 Management Shell.

#### Additional Requirements For Exchange2010

- The Tools4ever Powershell Agent Service must be installed on a computer that has the "Windows Management Framework Core" installed. This is allowed to be a different computer than the one that runs the UMRA service.
- Scripting must be enabled in the powershell environment on the computer that runs the PS Agent service. To do so, open a powershell command prompt and issue the command "SetExecutionpolicy Remotesigned"
- The computer on which the Powershell Agent Service is installed must be able to connect by remote powershell to your Exchange2010 environment.

#### Resources

The Windows Management Framework Core can be found here:  
<http://support.microsoft.com/kb/968929>

For all detailed information about Exchange2010 and Powershell, see <http://technet.microsoft.com/en-us/library/bb124558.aspx>

### 3.11.2. Accessing Exchange 2010 functionality from an UMRA project

#### How to Access Exchange2010 functionality from within an UMRA script

To access 2010 functionality from within a UMRA script perform the following script actions in order:

- Setup a new Powershell Agent Service Session with the with the UMRA script action "Setup Powershell Agent Service Session".

This action generates a clean Powershell runtime environment within the Tools4ever Powershell Service. It returns the ID of the Powershell Agent service session that is to be used as input parameter for all Exchange actions.

- Connect this environment to an Exchange 2010 host by means of the UMRA script action "Setup Exchange Session (Exchange 2010)".

This action creates a Remote Powershell connection (PSSession) to the Exchange Powershell module on the Exchange server. The "Connection URI" parameter specifies the exact module to load. For a regular Exchange2010 server this is "http://<Exchange 2010 server name (dns format) >/Powershell.

- Next use any sequence of Exchange2010 and other UMRA script actions as required. If any of the other actions also uses Powershell functionality, make sure that they use a different Powershell Agent service session.
- When finished using exchange2010, use "Close Exchange Session (exchange 2010)" to disconnect the Powershell session from exchange.  
Do not omit this action, it is required to release used resources.

Note that this action is generally used immediately before closing the hosting Powershell Agent session.

- Next use the script action "Release Powershell Agent service session", to close the powershell environment.



Note that generally only one Exchange session to a single Exchange 2010 host is required, and the entire forest can be managed from that connection. If more exchange 2010 hosts should be connected for any reason, a separate Powershell Agent Service Session should be used for each connection.

Also note that the same Powershell Agent Session can be used in different sequential umra projects, without the need to close the session in each individual project.

### **How to connect to other providers than Microsoft Exchange2010 server**

In addition to a genuine Microsoft Exchange 2010 server, there are a several other applications that support management by remote powershell. If these remote powershell environments (modules) expose the same powershell command-lets and parameters (or a relevant subset thereof) as Exchange 2010 server, then they too can be accessed by the UMRA Exchange 2010 actions.

To connect to such an environment, change the Connection URI in the "Setup Exchange Session" command to the powershell module provided by the application.

Note that if an access denied error is logged that refers to a specific powershell parameter, this often means that the particular parameter is not exposed by the remote powershell module. Often setting the associated UMRA action property to "not specified" alleviates this issue.

### **Outlook Live**

An example of such an application is Outlook Live. If you have administration rights in your Outlook Live environment, you can use the "Setup Exchange session" to connect to your Outlook Live domain.

Use "outlooklive" as value for the Connection URI,(or specify <https://ps.outlook.com/powershell> directly) and specify the correct username and Password of your administrative Outlook Live account.

After a successful connection it is possible to use the exchange 2010 actions. Note that only a subset of the actions are supported by Outlook Live. Generally only those actions are supported that directly relate to accounts and mailboxes (like "create user and mailbox", and "Edit mailbox" etc.).

### **3.12. Office 365**

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

## C H A P T E R

### 3.12.1. Introduction Office 365

In this and the following topics you can find general information how to use Office 365 actions in UMRA.

#### General

UMRA communicates (through the Tools4ever Powershell Service) with the Office 365 environment. The UMRA actions are as close as possible to the individual Powershell commands available in the Office 365 MS Online module.

#### Additional Requirements For Office 365

- The Tools4ever Powershell Agent Service must be installed on a Windows 2008 R2 or Windows 7 computer.
- The computer on which the Powershell Agent Service is installed must be able to connect the Office 365 environment.
- The Office 365 cmdlets.

#### Resources

For all detailed information about how to install the Office 365 cmdlets, see <http://onlinehelp.microsoft.com/office365-enterprises/hh124998.aspx>

### 3.12.2. Office 365 Users

#### Licensing users

When adding users, they are not licensed within Office 365 by default.

Use the **License assignment** property in the **Office 365 Create user** action or use the **Office 365 Edit user license** action. Licenses can be obtained with the **Office 365 List account SKUs** action. This action

produces a table with all licenses. The **AccountSkuld** column contains the information to be used when editing a user license. This is the fourth column (so its index is 3, because the index starts at column 0.)

### Check for existence

Use the **Office 365 Get user exists** action to check if a user exists. Do not use the **Office 365 List users** action with the **Object id** or **User principal name** property to check if a user exists, because it will throw an error when a non-existing user principal name or object id are specified.

## 3.13. Powershell Agent service

This document describes the Powershell Agent Service. The Powershell Agent Service is part of the UMRA software and enables Powershell support for UMRA projects.

### Concept

The Powershell Agent Service is a piece of software that can execute Powershell commands and scripts. The Powershell Agent Service has an interface that communicates with the UMRA software. Both the UMRA Console and UMRA Service application can send Powershell scripts to the agent service. The agent service checks the access rights of the caller and executes the Powershell script if access is granted. The data that is generated by Powershell can be returned to UMRA.

The Powershell Agent Service is a service application: it runs always as a separate process and the service has no graphical user interface. The Powershell Agent Service heavily uses the Microsoft .NET framework. Therefore, some additional requirements apply for the computer that runs the agent service.

In UMRA, additional UMRA actions are available that deal with Powershell. When an UMRA project contains these types of actions, Powershell scripts are sent to the Powershell Agent Service when UMRA executes the project. For each action, UMRA sends a single Powershell script to the service. The conversion from an UMRA action to a Powershell script is well defined according to a set of rules. For each UMRA action that uses Powershell functionality, an UMRA dynamic action file exists. This file (XML) defines how the Powershell script must be composed and depends on the UMRA action properties. In other words: the dynamic action file specifies both the UMRA action and the Powershell script and how these two items are related.

### 3.13.1. Powershell Agent service

To support Powershell, UMRA uses Tools4ever's Powershell Agent service. The Powershell Agent service is able to execute Powershell scripts. Different scripts can be used for example to create, manage and delete mailboxes and other Exchange resources. The Powershell Agent service operates as follows:

1. An application (UMRA) sends a Powershell script to the Powershell Agent service;
2. The Powershell Agent service checks the access rights of the account that offers the script;

3. If access is granted, the Powershell Agent service sets up the Powershell runtime environment or uses an existing Powershell Agent service session;
4. The Powershell Agent executes the script.
5. Result objects generated by the Powershell script can be captured and send back to the calling application.

### **3.13.2. UMRA - Powershell Agent service**

To operate with the Powershell Agent service, UMRA communicates with the Powershell Agent service. When an UMRA project script is executed, one or more UMRA actions may contain Powershell (Exchange 2007) functionality. For these actions, UMRA generates a Powershell script. The script is then send to the Powershell Agent service. The Powershell Agent service executes the Powershell script and returns any result back to UMRA.

### **3.13.3. UMRA action - Powershell script conversion**

A number of actions in UMRA deal with Powershell. For instance, all the actions to manage Exchange 2007 mailboxes. For these actions, UMRA generates a Powershell script. For each of these actions, a single Powershell script is generated. The content of the Powershell script is determined by the action itself and the specification of the UMRA properties. With this information, UMRA converts the UMRA action specification into a Powershell script.

### **3.13.4. UMRA dynamic actions**

A number of actions in UMRA deal with Powershell. For instance, all the actions to manage Exchange 2007 mailboxes. For these actions, UMRA can generate a Powershell script. The contents of the Powershell script is determined by the UMRA action and the UMRA action property values. The conversion is straightforward and according to a number of rules. The exact conversion is described in a document that is available for each action. This document has an XML format. The XML-document contains the complete specification of the UMRA action, including descriptions, properties, property dependencies, the Powershell script 'template' and the conversion to generate the script.

To add a new action that uses Powershell to UMRA, a new XML-document must be created. With such document, describing the UMRA action in detail, UMRA can be extended with the new action. To add the new action, no new version of UMRA is required. Instead, the XML document must be imported to UMRA. These type of actions are referred to a **UMRA dynamic actions**.

### **3.13.5. Installation**

The Powershell Agent service is a special piece of software that is used to execute Powershell commandlets. Technically, the Powershell Agent service is a Powershell host that is able to execute Powershell commandlets like the Windows Powershell command-line environment and the Exchange Powershell Console application.

The Powershell Agent service is part of the UMRA software and shipped as part of the UMRA software. The Powershell Agent service is completely installed and managed from within the UMRA Console application. To install and manage the Powershell Agent service, the UMRA Console application must be installed first.

### **Powershell Agent service setup - Requirements**

The following requirements apply to setup the Powershell Agent service:

1. The Powershell Agent service can be installed on 32-bit and 64-bit machines. The Powershell Agent service is installed from the UMRA Console application. The application will detect automatically if the target computer is 32- or 64-bit based.
2. In order to manage Active Directory / Exchange 2007, there must be at least one domain controller running Active Directory on Windows 2003 or Windows 2008 (in a network with only Windows 2000 domain controllers, the service will not be able to manage Active Directory).
3. In order to manage Exchange 2007, the Microsoft Exchange 2007 Management software must be installed on the computer that is going to run the Powershell Agent service. The Microsoft Exchange 2007 Management software is part of the Microsoft Exchange 2007 software. The software is available for both 32-bit and 64-bit computers. As part of the installation of the Microsoft Exchange 2007 Management software, a number of other software components are installed and configured, for instance the .NET framework. Note that the Microsoft Exchange 2007 Management tools software is not part of the UMRA software. For more information to install the Microsoft Exchange 2007 Management software on a 32-bit platform, see *Setting up the Exchange 2007 Management Tools on a 32-bit platform* for more information.  
**In case the Powershell Agent service is installed to manage other systems than Exchange 2007, for instance SAP, the Microsoft Exchange 2007 Management software does not need to be installed on the computer. Instead, Windows Powershell software and the .NET Framework must be installed separately in this case. This software is available at no charge from Microsoft's web-site.**
4. In order to manage Exchange 2007, the Powershell Agent service computer must be a member of the Active Directory domain of the Exchange server.
5. The UMRA Console application must be installed on a computer (any other computer of the domain) to install the Powershell Agent service.

### **Powershell Agent service setup - Procedure**

To setup the Powershell Agent service, perform the following steps:

1. Logon to the computer that has the UMRA Console application installed with administrative account. The account must have sufficient access to install a service on the computer that is going to run the Powershell Agent service. By default, an account that is a member of the **Administrators** or **Domain Admins** group normally has sufficient access rights;
2. Start the UMRA Console application.

3. Select menu option **Tools, Options** and select the window **Powershell Agent**. The following dialog will appear:

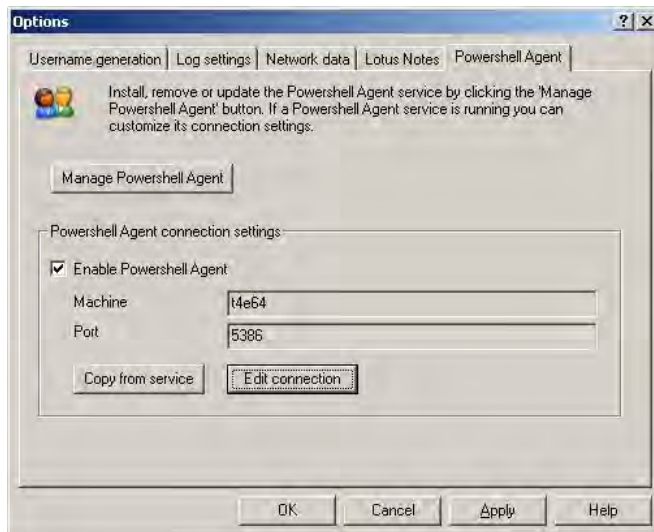


Figure: Manage Powershell Agent service dialog.

For more information about the **Manage Powershell Agent service dialog**, go to *Powershell Agent connection settings* on page 5.

Click button **Manage Powershell Agent**. A wizard is shown to install the **Powershell Agent** service. Go to *Powershell Agent service wizard - Manage* on page 8 to read about the steps of the Powershell Agent service installation wizard.

### Powershell Agent connection settings

To successfully execute UMRA actions with Powershell functionality, the UMRA application must be connected to the Powershell Agent service. The UMRA application is either the UMRA Service or the UMRA Console application. For mass projects, the UMRA application is the UMRA Console. For all other UMRA projects, the UMRA application is the UMRA Service.

For both the UMRA Console and UMRA Service application, the **Powershell Agent service** connection needs to be configured with the **Powershell Agent connection settings** dialog. Use the following instructions to get to the **Powershell Agent connection settings**.

**UMRA Service:** From the UMRA Console application, connect to the UMRA Service and select menu option **UMRA Service, Service properties....** Select window **Powershell Agent**.



**UMRA Console:** Select menu option **Tools, Options...** Select windows **Powershell Agent**.

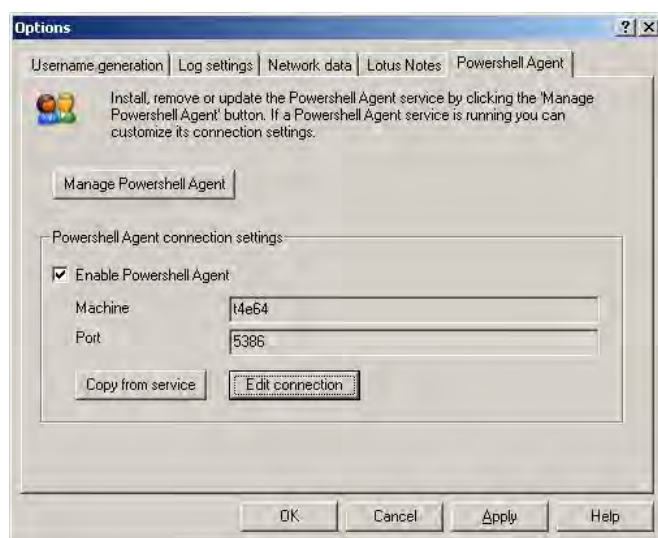


Figure: Powershell Agent connection settings dialog.

Use the **Manage Powershell Agent** button to install, update or delete the Powershell Agent service. For more information about the Manage Powershell Agent wizard, go to *Powershell Agent service wizard - Manage* on page 8.

Enable the Powershell Agent connection settings by checking the **Enable Powershell Agent** check box. UMRA will now use these connection settings to connect to the Powershell Agent service. If there are no machine and port number specified yet, click the **Edit connection** button and enter the connection settings.

Note that you must specify a machine name that is already running the Powershell Agent service and that you must enter the correct port number.

By choosing **OK** or **Apply**, this dialog will check these settings and the UMRA log will show information of the settings that are applied for the Powershell Agent service.

If the Powershell Agent connection settings are configured on the UMRA Service, these settings can be copied to the UMRA Console by clicking the **Copy from service** button and vice versa. To definitely copy the settings, confirm the following dialog:

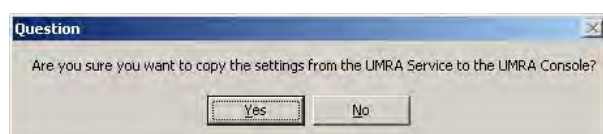


Figure: Confirm copying the Powershell Agent service connection settings

Edit the existing setting by clicking the **Edit connection** button. A dialog will appear to edit the connection settings. For more information about this dialog, go to *Edit connection settings* on page 7.

If only the UMRA Service executes project scripts with Powershell functionality, there is no need to setup the Powershell connection settings for the UMRA Console.

#### Powershell Agent service - Edit connection settings

Change the Powershell Agent connection settings with this dialog. To browse for a machine name, click the browse button. Otherwise, specify the **machine** name that runs the Powershell Agent service.

In the **Port** edit box, enter the port number which the Powershell Agent service is using.



Figure: Edit the Powershell Agent service connection settings

Click **OK** to apply the new settings.

### Powershell Agent service wizard - Manage

Use this wizard to install the Powershell Agent service on a computer, update the Powershell Agent service or delete the Powershell Agent service.



Figure: Welcome to this wizard to manage the Powershell Agent service

Choose **Install or upgrade the Powershell Agent service** if the Powershell Agent service is not installed already or needs an update. Choose **Delete the Powershell Agent service** if you want to delete the Powershell Agent service. Click **Next**.

### Powershell Agent service wizard - Specify server

Specify a server for the Powershell Agent service wizard to run on. If the Powershell Agent service already runs on that server, the Manage Powershell Agent service wizard will update the service. For more information about updating, go to *Powershell Agent service wizard - Update Powershell Agent service* on page 12.

Note that the server must meet the PowerShell Agent service installation requirements. For more information about installation requirements, go to *PowerShell Agent service setup - Requirements* on page 4.



Figure: Specify the name of the server to install the PowerShell Agent service on.

Click the **Use local computer** to get the name of the local computer. Click the browse button to open a **Select computer** dialog to browse for another server. Click **Next**.

#### PowerShell Agent service wizard - Specify port number

Specify the port number for the Powershell Agent service.

Make sure the port specified is not already in use.



Figure: Specify the port for the Powershell Agent service

### Powershell Agent service wizard - Specify service directory

Specify the Agent service directory on the target computer. The directory must be specified as a path, including the logical disk of the target computer. The logical disk must exist. The directory is created if it does not exist. By default, the wizard determines and shows the default directory.

Note that the server must meet the PowerShell Agent service installation requirements. For more information about installation requirements, go to *PowerShell Agent service setup - Requirements* on page 4.

### Powershell Agent service wizard - Specify account

Specify a user account with this dialog. This user account will be used by the Powershell Agent service. If the account does not exist it will be created with the specified password. If the account does not exist, the correct password must be specified in order for the Powershell Agent service to be able to log on with that account.



Figure: Specify the Powershell Agent service account

The **Account name** field displays the account that the Powershell Agent service will use. The **Password** field and the **Confirm password** field are used to create the account, if it does not exist already.

Use the browse button to search a specific account. Click **Next**.

### Powershell Agent service wizard - Specify account group

Specify the group of which the Powershell Agent service account must be a member of. If the user account will be created, it will be a member of this group. If the user account already exists, but is not a member of the specified group, the wizard will add the user to this group.



Figure: Specify the security group for the service account

Use the browse button to browse to the group. Click **Next** to continue.

### PowerShell Agent service wizard - Specify user account group

Specify the group that the user account, which is accessing the PowerShell Agent service, must be a member of. When using UMRA Console mass projects, the account is the logged on user. With UMRA Automation and Form projects, this user is the UMRA Service account.

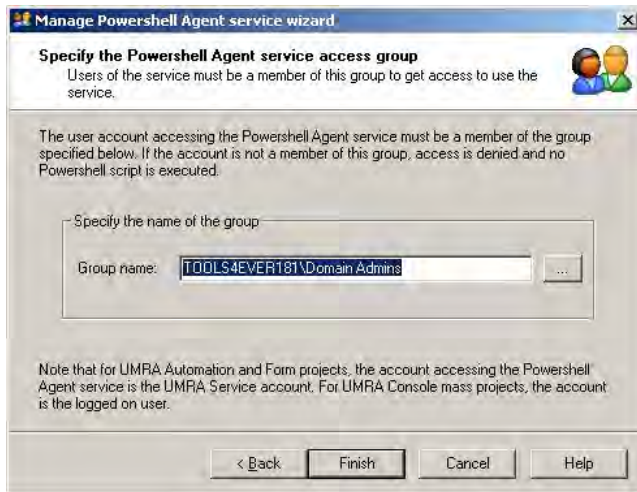


Figure: Specify the PowerShell Agent service access group

Use the browse button to browse to the group. Click **Finish** to install the PowerShell Agent service.

### PowerShell Agent service wizard - Update PowerShell Agent service

When the PowerShell Agent service is already installed on the specified server, the following dialog will appear:

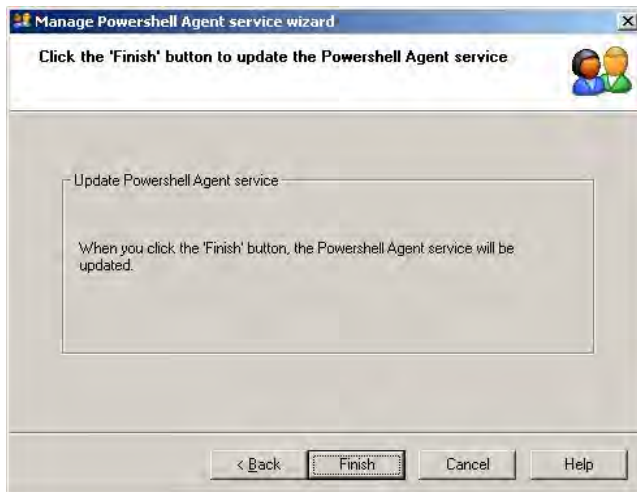


Figure: Update the PowerShell Agent service

The PowerShell Agent service currently installed on the specified machine runs an older version than the one available with this UMRA version. Therefore, this wizard will stop the currently running PowerShell Agent service when you click the **Finish** button. Then the wizard will update the PowerShell Agent service and start the service again.

Note that the PowerShell Agent service needs to run the same version number as the version number the UMRA Console is running.



### Powershell Agent service wizard - Delete all files

When the Powershell Agent service is deleted, some files, in the Powershell Agent service directory will remain. This includes log files and configuration files. If these files also need to be removed, select the **Delete all files found in the Powershell Agent service directory and the directory itself** check box. As the check box says, the wizard will now remove the entire Powershell Agent service install directory.

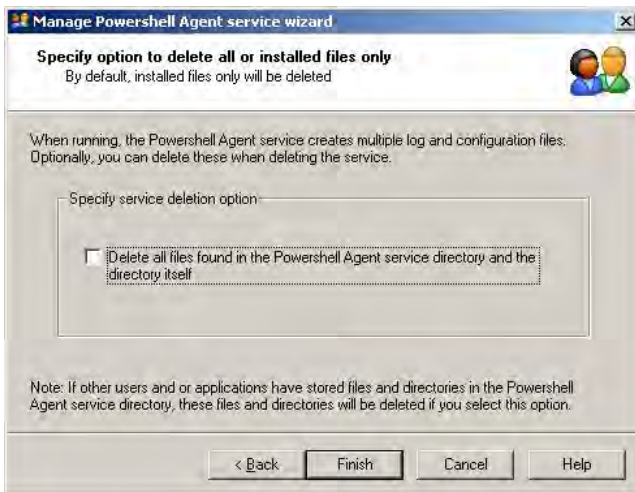


Figure: Option to delete the Powershell Agent service installation directory

### Manual installation of the Powershell Agent service

The Powershell Agent service is best installed using the UMRA Console application. If it is not possible to install the service using the UMRA Console, the following procedure can be used to install the Powershell Agent service. In this procedure the UMRA Powershell Agent service is installed on a computer running Windows XP (32-bit) that is not a member of a domain. When running Windows 2003 or a 64-bit OS, a similar procedure applies.

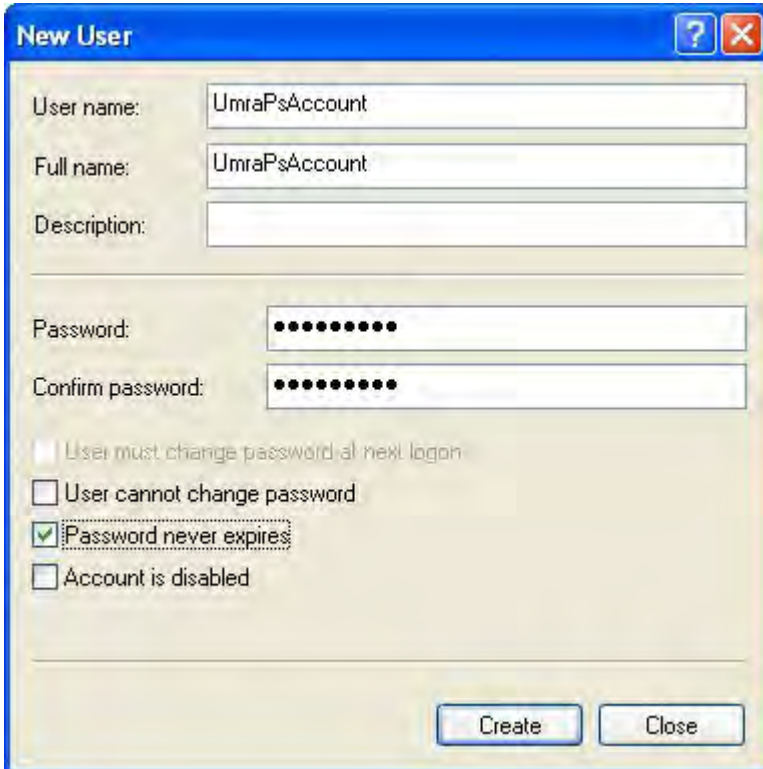
1. **Prerequisite: .NET Frame 2.0**  
The .NET Framework 2.0 must be installed on the computer. When the latest updates are applied to the XP machine, this is automatically the case. the .NET Framework 2.0 is available from the Microsoft web-site.
2. **Prerequisite: Windows Powershell 1.0**  
The computer must have Powershell 1.0 installed. Windows Powershell 1.0 is available from the Microsoft web-site, both for 32- and 64-bit systems.
3. **Prerequisite: UMRA Console**  
Install the UMRA Console application on the computer.
4. **Program directory**  
Create a directory on the computer that is used by the Powershell Agent service. Default:  
C:\Program Files\Tools4ever\PowerShellAgent.

5. **Create subdirectories**  
In the new directory, create subdirectories **Log**, **SnapIns** and **PowershellAgentLib**. On a 32-bit system, create sub-directory **SAP** in directory **SnapIns**.
6. **Setup subdirectory Microsoft.VC80.CRT**  
Locate the UMRA Console directory and navigate to subdirectory **SvcSetup**. Default: **C:\Program Files\Tools4ever\User Management Resource Administrator\SvcSetup**. The directory contains subdirectory **amd64** for 64-bit systems and **x86** for 32-bit systems. Copy the contents of the appropriate directory to the program directory created in step 4. For both cases, the name of the directory is **Microsoft.VC80.CRT**.
7. **Setup subdirectory PowershellAgentLib**  
From the **SvcSetup** directory, copy file **PowershellAgentLib.xmllib** to subdirectory **PowershellAgentLib**.
8. **Setup subdirectory SnapIns (32-bit systems only)**  
On 32-bit systems, copy the contents of **SvcSetup** subdirectory **SnapIns\SAP** to **SnapIns\SAP**.
9. **Copy Powershell Agent service binary file**  
From the **SvcSetup** directory, copy file **T4ePowerShellAgentX64.exe** (64-bit systems) or file **T4ePowershellAgentW32.exe** (32-bit systems) to the program directory created in step 4.
10. With a editor, like notepad, create a text file, **PsAgentSettings.xml** in the program directory. The xml file should contain the following contents:

```
<AgentConfiguration>
  <ExecAccessGroup>Administrators</ExecAccessGroup>
</AgentConfiguration>
```

Members of the specified group are granted access rights to configure the Powershell Agent service.

11. **Service account**  
On XP, start **Control Panel, Administrative Tools, Computer Management**. Select **Local Users and Groups**. Right click **Users** and select menu option **New User...** Specify the name and attributes for the account. Remember the password and check the option **Password never expires**.



**New User**

User name: UmraPsAccount

Full name: UmraPsAccount

Description:

Password: .....

Confirm password: .....

☐ User must change password at next logon

☐ User cannot change password

☒ Password never expires

☐ Account is disabled

Create Close

12. **Administrative group membership**

Right click the new account from the list of accounts and select menu option **properties**. Select tab **Member Of** and add the new account to the group **Administrators**.

13. **User right assignment - Log on as a service**

On XP, start **Control Panel, Administrative Tools, Local Security Policy**. In the **Security Settings** tree, select **User Rights Assignment**. In the list on the right side, select policy **Log on as a service**. Select menu option **Properties, Add User or Group** and add the UMRA Powershell Agent service account created in step 5.



#### 14. Create the Powershell Agent service

Start Powershell and use commandlet **New-Service** to create the service. Specify the commandlet parameters according to the following table:

Parameter	Description	Example
-name	The name of the service, must be T4ePowershellAgent	T4ePowershellAgent
-binaryPathName	The name, including the path, of the service executable. Note that the name differs for 32-bit and 64-bit systems.	C:\Program Files\Tools4ever\PowerShellAgent\T4ePowershellAgentW32.exe or C:\Program Files\Tools4ever\PowerShellAgent\T4ePowershellAgentX64.exe
-displayName	Tools4ever's Powershell Agent service	Tools4ever's Powershell Agent service
-startupType	Automatic	Automatic
-credential	The name and password of the UMRA Powershell Agent service account, as created in step 11.	UmraPsAccount BigSecret0673

When the service is not installed by specifying single command line, the **New-Service** commandlet will create the service when the **-name** and **-binaryPathName** are specified. Next, use **Control Panel, Administrative Tools, Services** to further configure the service. The service display name is not important but you must specify the account created in step 11, using tab **Log On** for the service. Do not start the service yet.

### 15. Setup the registry

Setup the required registry keys of the Powershell Agent service. The only required key is **ExecAccessGroup**. Refer to topic Registry settings for more information.

The service is now installed. Use **Control Panel, Administrative Tools, Services** to start the service.

### 3.13.6. Configuration and settings

The Powershell Agent Service uses a number of settings and configuration items to function. These are described in topics of this chapter.

#### Access and Security

To access the Powershell Agent Service, users must be a member of a group. This group is specified when the service is setup. When a user accesses the Powershell Agent Service, the service checks if the user is a member of this security group. If this is the case, access is granted and the requested Powershell script is processed and executed by the agent service. If the user account is not a member of the group, access is denied and the requested Powershell script is not executed.

To modify the group when the service is already setup, you will need to update the registry and restart the service. See registry key **ExecAccessGroup** in Registry settings for more information.

The Powershell Agent Service is either accessed by the UMRA Console or UMRA Service application. With the UMRA Console application, the user account that accesses the Powershell Agent Service is the user that runs the UMRA Console application. For the UMRA Service application, the user account accessing the Powershell Agent Service is the service account of the UMRA Service.

#### Licensing

To execute a Powershell script through the Powershell Agent Service, a separate UMRA license is required. Without the correct license, the UMRA software will not execute an action that uses Powershell functions. Since Exchange 2007 actions use Powershell functions, this is also true for the UMRA Exchange 2007 actions.

Contact your UMRA reseller to obtain a license that support Exchange 2007 and Powershell.

### **Log information**

The Powershell Agent Service writes log information to a cyclic log file. The file is located in the **Log** subdirectory of the service program directory. The default location is

C:\Program Files\Tools4ever\PowerShellAgent\Log.

The log file contains general progress information and all errors that occur.

The log file name have the format:

[log directory][log file label][log cycle sequence number].txt

By default, the log files are a set of 10 files with cycle numbers 000, 001, ... , 009 and label T4ePsLog. Each file has a maximum size of 5MB. When file is full, the next file is generated. When all files are full, the first file is emptied. The resulting files are: T4ePsLog000.txt, T4ePsLog001.txt, ... , T4ePsLog009.txt. To change the log file settings, you need to update the log related registry settings.

### **Powershell snap-ins**

Windows PowerShell snap-ins provide a mechanism for registering sets of cmdlets and providers with the shell, thus extending the functionality of the shell. If a PowerShell script uses a cmdlet that is registered in a specific snap-in, the snap-in needs to be registered (installed) on the machine that executes the PowerShell script, e.g. the computer that runs the PowerShell Agent service. If the snap-in is not registered, PowerShell will no be able to execute the cmdlet. In most cases, groups of cmdlets with related functionality are assembled in the same snap-in.

Example: by default, the cmdlets that deal with Exchange 2007 are not available by default in PowerShell. Instead, the cmdlets are defined in the snap-in

#### **Microsoft.Exchange.Management.PowerShell.Admin**

To execute scripts that use cmdlets to manage Exchange 2007, this snap-in needs to be installed on the computer that runs the PowerShell Agent service. The snap-in is installed as part of the installation procedure of Exchange server 2007, or the Exchange 2007 Management tools.

To execute PowerShell scripts that require a certain snap-in to be installed with UMRA, the XML-file specification of the dynamic action must specify the snap-in in the configuration section of the XML-file. See *Configuration section* (on page 27) for more information.

### Registry settings

The Powershell Agent Service uses several registry settings for security, logging settings etc. These settings are set to the default values automatically when the service is installed. If changed later manually, the agent service must be restarted for the new settings to take effect.

The Powershell Agent Service uses the following registry settings:

<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Config
<b>Name</b>	ExecAccessGroup
<b>Type</b>	Text
<b>Initialization</b>	Powershell Agent Service installation
<b>Example</b>	Domain Admins
<b>Description</b>	The user who accesses the Powershell Agent Service in order to execute a Powershell script must be a member of this group. If not, access is denied and no Powershell script is executed.

<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Config
<b>Name</b>	SessionTimeToLive
<b>Type</b>	DWORD
<b>Initialization</b>	Manual (default value, equal to value used when not specified: 240)
<b>Example</b>	330
<b>Description</b>	The time interval in minutes that specifies the maximum idle time of a <i>Powershell Agent service session</i> on page 78. When a Powershell Agent service session remains idle for a longer period, it is automatically released. The idle interval is interrupted when the session is used by an UMRA client, for instance when a dynamic action is executed in the context of the session. When the idle interval is interrupted, the idle time is reset to zero. The default value is 240 minutes (4 hours). when the setting is updated, the Powershell Agent service needs to be restarted to make the setting effective. The value used is written to the log when the service is started.

<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Config
<b>Name</b>	Updated

<b>Type</b>	DWORD
<b>Initialization</b>	Powershell Agent Service installation
<b>Example</b>	0 or 1
<b>Description</b>	Internal use only: When the service reads a value of 0 during startup, the service reads registry initialization values from file PsAgentSetting.xml. Next, the value is set to 1.

<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Log
<b>Name</b>	BaseDirectory
<b>Type</b>	Text
<b>Initialization</b>	Manual
<b>Example</b>	D:\UMRA\PsAgentLog
<b>Description</b>	The base directory of the cyclic log file sequence of the Powershell Agent Service. If not specified, the service sets this value to the subdirectory <b>Log</b> in the service program directory.

<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Log
<b>Name</b>	FileLabel
<b>Type</b>	Text
<b>Initialization</b>	Manual
<b>Example</b>	T4ePsLog
<b>Description</b>	The name of a single log file, without the directory part, cycle number and file extension. If not specified,

<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Log
<b>Name</b>	CycleCount
<b>Type</b>	DWORD
<b>Initialization</b>	Manual
<b>Example</b>	10
<b>Description</b>	The number of files contained in a full log file cycle. When all logfiles are filled, the first file (cycle 0) is removed and recreated.



<b>Key</b>	HKLM\SYSTEM\CurrentControlSet\Services\T4ePowershellAgent\Log
<b>Name</b>	CycleKBytes
<b>Type</b>	DWORD
<b>Initialization</b>	Manual
<b>Example</b>	5120
<b>Description</b>	The size of a single log file in KBytes.

### 3.13.7. UMRA dynamic actions

All UMRA actions that use Powershell are dynamic actions. The action is called dynamic since it is not part of the native UMRA software code. Instead, an XML-file specifies the action. This file is imported into the UMRA dynamic action library. This document describes the format of the XML-file, the dynamic actions library and an example.

#### XML file specification

The XML file that specifies an UMRA dynamic Powershell action can be divided into the following sections:

1. Basic section: Specifies of some default characteristics of the action;
2. Properties section: The UMRA action properties;
3. Script section: The specification of the Powershell script and how the script depends on the UMRA properties.

To create such a file, you can use any editor, including notepad. Some of the more advanced editor are better equipped to create XML files and show for instance if the file contents is well-formed. This document describes these parts in detail. An example of a complete XML file is shown below.

```
<?xml version="1.0" encoding="UTF-16"?>
<UmraDynamicAction version="1" type="Powershell">
  <General>
    <Name>Restart a service</Name>
    <Description>Restart a particular service on the current
computer.</Description>
    <ActionTreeLabels>Powershell, Example actions</ActionTreeLabels>
    <ActionImage>17</ActionImage>
    <Version>101</Version>
  </General>
  <ActionProperties>
```

```
<ActionProperty>
  <DisplayName>Service name</DisplayName>
  <Name>ServiceName</Name>
  <Description>The name of the service to restart.</Description>
  <ValueType>Text</ValueType>
</ActionProperty>
</ActionProperties>
<Script>
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>
</Script>
</UmraDynamicAction>
```

#### *Basic section*

The basic section includes the following xml elements in the dynamic action specification file:

1. Declaration
2. General
3. Configuration

#### Declaration

The XML file that specifies an UMRA dynamic action, should start as follows:

```
<?xml version="1.0" encoding="UTF-16"?>
```

This declaration includes the UTF-16 encoding declaration. Do not use another encoding method. For UMRA dynamic XML files, no XML schema, XML Schema Definition (XSD) or Document Type Definition (DTD) is available.

Next, the UmraDynamicAction element specifies the type of dynamic action: Powershell.

```
<UmraDynamicAction version="1" type="Powershell">
```

The element `UmraDynamicAction` encloses the entire dynamic action specification, including the **General**, **ActionProperties** and **Script** sections.

#### General section

The general section specifies a number of properties of the UMRA dynamic Powershell action.

<b>Element:</b>	General
<b>Parent element:</b>	UmraDynamicAction
<b>Mandatory:</b>	Yes
<b>Attributes:</b>	
<b>Description:</b>	The outer element of the general section of the UMRA Powershell dynamic action specification.

The General element contains the following child elements:

<b>Element:</b>	Name
<b>Parent element:</b>	General
<b>Mandatory:</b>	Yes
<b>Attributes:</b>	-
<b>Description:</b>	The name of the dynamic action. The name identifies the action and should be unique. If a new dynamic action is imported with the same name as an action that already exists, the existing action is overwritten with respect to the version and signature information.

<b>Element:</b>	Description
<b>Parent element:</b>	General
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	The description of the dynamic action. The description is shown when the user positions the mouse on top of the action in the UMRA action tree. The description should describe the action in general, not all of the properties.

<b>Element:</b>	ActionTreeLabels
<b>Parent element:</b>	General
<b>Mandatory:</b>	Yes
<b>Attributes:</b>	-
<b>Description:</b>	The specification of the folder of the action in the UMRA action tree. You can position the action in any folder, also in a new folder. The folder is specified as a comma separated string. Start with the parent folder, the child folder etc. If a folder does not already exist, it is created automatically. Example: To position the action in folder <b>Test</b> of parent folder <b>My actions</b> , specify: <b>My actions, Test</b> .

<b>Element:</b>	ActionTreePosition
<b>Parent element:</b>	General
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	The relative position in the UMRA action tree. If a folder in the UMRA action tree contains multiple actions, the actions are sorted based on this number. Values can range from 0 to 1000000. To allow new actions added later between existing actions, it is recommended to use a step value of 100 or 1000 between subsequent actions (e.g: 1000, 2000, 3000 etc)

<b>Element:</b>	ActionImage
<b>Parent element:</b>	General
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	The index of the image to be associated with the action. The available images correspond with the images used for rows in UMRA form tables. Subtract 1 from the index shown in the figure below.

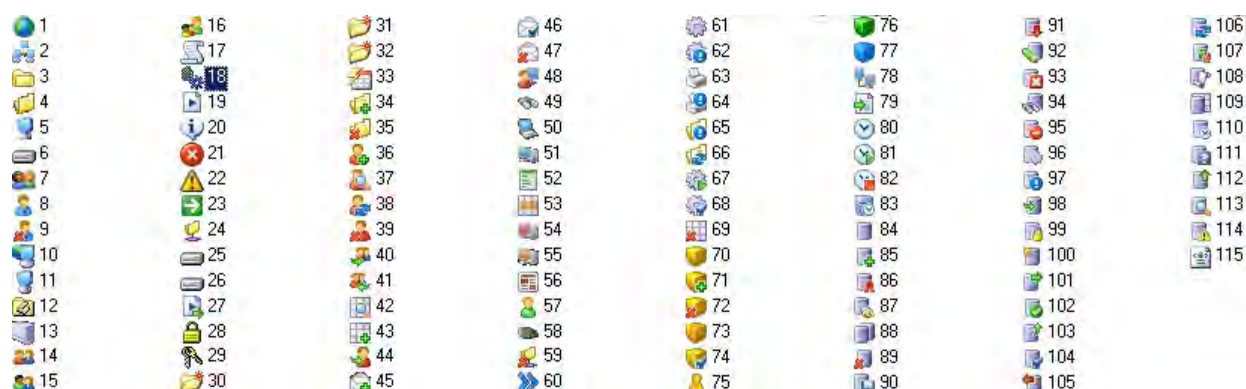


Figure: The possible images of a dynamic action. Subtract 1 from the number shown to obtain the correct index.

<b>Element:</b>	Obsolete
<b>Parent element:</b>	General
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	If set to a non-zero value (for example 'Yes' or 1) the action is obsolete and not shown in the UMRA actions tree. If the element is omitted or set to a value of 0 or 'No', the action is not obsolete and shown in the UMRA actions tree. Even if the action is obsolete, it is still executed if it is contained in an UMRA script that was created earlier.

<b>Element:</b>	Version
<b>Parent element:</b>	General
<b>Mandatory:</b>	Yes
<b>Attributes:</b>	-
<b>Description:</b>	<p>The version number of this action. The version number is used to allow upgrades of the same action (e.g. the action with the same name). Possible values range from 1 to 100000. Normally, an action starts with a version number of 1. Each time you update the XML file, the version number should be incremented.</p> <p>An UMRA dynamic action is upgraded by changing the contents of the XML file and then reloading the file. The reload operation only has effect if the version number of the action exceeds the version number of the same action to be upgraded. See <i>Upgrading dynamic action</i> on page 63 for more information.</p>

<b>Element:</b>	HelpId
<b>Parent element:</b>	General
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	A number used by UMRA dynamic actions that are created by Tools4ever. The number refers to the online help file.

<b>Element:</b>	Signature
<b>Parent element:</b>	General
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	A text string representing the signature of the action. If the signature is present, the action cannot be upgraded then by authorized UMRA personnel. This is to protect the 'official' UMRA dynamic actions from accidental changes. See <i>Signature of dynamic actions</i> (see "Signature of UMRA dynamic actions" on page 63) for more information. If a signature is present, do not update the file since you will not be able to reload and upgrade the file.

#### Configuration section

The configuration section specifies the configuration settings that deal with either the UMRA software, the Powershell Agent Service or both. The element is defined as follows:

<b>Element:</b>	Configuration
<b>Parent element:</b>	UmraDynamicAction
<b>Mandatory:</b>	No
<b>Attributes:</b>	
<b>Description:</b>	The outer element of the configuration section of the UMRA Powershell dynamic action specification. If the specification does not contain any configuration items, the default values apply and the element can be omitted.

The **Configuration** element can contain the following child elements:

<b>Element:</b>	PowershellSnapIn
<b>Parent element:</b>	Configuration
<b>Mandatory:</b>	Depends on cmdlets used in script.
<b>Multivalue:</b>	Possible, for each snap-in, a separate element must exist.
<b>Attributes:</b>	-
<b>Description:</b>	<p>The name of the snap-in. The snap-in is loaded at run-time by the Powershell Agent Service. A snap-in contains the code of cmdlets. If the snap-in is not specified, the cmdlets of the snap-in cannot be used in a Powershell script. Example: All Exchange 2007 specific cmdlets are specified in snap-in <b>Microsoft.Exchange.Management.PowerShell.Admin</b>. See <i>Powershell snap-in</i> (see "Powershell snap-ins" on page 19) for more information.</p> <p>To specify multiple snap-ins, create multiple <b>PowershellSnapIn</b> XML-elements.</p>

<b>Element:</b>	PowershellAgentLibrary
<b>Parent element:</b>	Configuration
<b>Mandatory:</b>	Depends on Powershell calls made to library in script.
<b>Multivalue:</b>	Possible, for each library, a separate element must exist.
<b>Attributes:</b>	-
<b>Description:</b>	<p>The name of the library to include. The library is maintained by the Powershell Agent Service. At runtime, the content of the library is loaded into the runspace of the Powershell Agent service thread that services the call.</p> <p>To specify multiple libraries, create multiple <b>PowershellAgentLibrary</b> XML-elements.</p>

<b>Element:</b>	ErrorHandling
<b>Parent element:</b>	Configuration
<b>Mandatory:</b>	No
<b>Attributes:</b>	propagate

<b>Description:</b>	The element has no contents, but only specifies the <b>propagate</b> attribute. The attribute is either <b>Yes</b> or <b>No</b> . The default value (not specified) is <b>Yes</b> . When set to <b>Yes</b> , the UMRA action will generate an error when an error occurs when the Powershell Agent service executes the script. Thus, the error that occurs at the Powershell Agent service side, propagates to UMRA and activates a defined error handler. If set to <b>No</b> , UMRA ignores any error that occurs at the Powershell Agent service side.
---------------------	--

<b>Element:</b>	Session
<b>Parent element:</b>	Configuration
<b>Mandatory:</b>	No
<b>Attributes:</b>	required
<b>Description:</b>	The element has no contents, but only specifies the <b>required</b> attribute. The attribute is either <b>Yes</b> or <b>No</b> . The default value (not specified) is <b>No</b> . When set to <b>Yes</b> , the UMRA action will execute in the context of a Powershell runspace. To identify the runspace, the action specifies property SessionID. The session is initialized with dynamic action Setup Powershell Agent service session.

#### *Properties section*

All the properties of the UMRA dynamic actions are specified in the xml specification file. The xml-element **ActionProperties** contains all properties as child elements.

Properties specification

ActionProperties

The ActionProperties section specifies the properties of the UMRA dynamic action. The element is defined as follows:

#### **ActionProperties**

<b>Element:</b>	ActionProperties
-----------------	------------------



<b>Parent element:</b>	UmraDynamicAction
<b>Mandatory:</b>	No, but normally, an UMRA dynamic Powershell action has properties.
<b>Attributes:</b>	
<b>Description:</b>	The outer element of all UMRA Powershell dynamic action properties. For each property, the element contains a child element <b>ActionProperty</b> .

The **ActionProperties** element contain a child element ActionProperty for each UMRA Powershell dynamic action property:

#### ActionProperty

<b>Element:</b>	ActionProperty
<b>Parent element:</b>	ActionProperties
<b>Mandatory:</b>	Yes, for each property
<b>Attributes:</b>	-
<b>Description:</b>	The base element for each UMRA Powershell dynamic action property. The element contains a number of child elements that completely define the property.

The **ActionProperty** element exists for each UMRA Powershell dynamic action property. It specifies the complete action property in a number of child elements:

#### Name

<b>Element:</b>	Name
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	Yes
<b>Attributes:</b>	-
<b>Description:</b>	<p>The name of the UMRA Powershell dynamic action property. The name defines the property. Hence, each property in the same action should have a unique name. The name is not shown in UMRA (instead the DisplayName) is shown. The name is used in the script section of the XML-file specification.</p> <p>It is recommend to use a simple name with no spaces for this element. Example:  <b>ExchangeServer</b> instead of <b>Exchange server</b>.</p>

**DisplayName**

<b>Element:</b>	DisplayName
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	<p>The display name of the UMRA Powershell dynamic action property. The name is shown in UMRA. If not specified, the value is set equal to the value of the element <b>Name</b>.</p> <p>It is recommend to use a user-friendly name for this property. Example: <b>Exchange server</b> instead of <b>ExchangeServer</b>.</p>

**Description**

<b>Element:</b>	Description
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	<p>The description of the UMRA Powershell dynamic action property. The description is shown in UMRA when the property is specified. To format the text shown by moving to the next line, use the escape sequence \n in the text. Example: First line\nNext line.</p>

**ValueType**

<b>Element:</b>	ValueType
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	<p>Optionally: the attribute encrypted="Yes" can be used. In this case, the input value must be text and it must be specified in UMRA using an encrypted variable. See <i>Encrypted properties</i> for more information.</p>

<b>Description:</b>	<p>The type of the property value. If not specified, the default value <b>Text</b> is used. The type is used to check the UMRA action input. When UMRA executes the action, it determines the value of the property. The type of the value found should correspond (or allow conversion) with the specified type. Possible values are:</p> <p>Text (also: String)</p> <p>Numeric (also: Number, Numerical, Int, Integer)</p> <p>LongInteger (64-bit integer, very large number)</p> <p>Boolean (also: Bool, Flag)</p> <p>Date-Time (also: Date, Time, DateTime)</p> <p>TextList (also: TextArray)</p> <p>Table</p>
---------------------	--

#### DefaultValue

<b>Element:</b>	DefaultValue
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	<p><b>type</b></p> <p>When specified, the type of the default value. If not specified, the same value as the value of element <b>ValueType</b> is used.</p>
<b>Description:</b>	<p>If specified, the default value of the property. The default value is generated when the action is added to an UMRA script. If the default value is a variable name, the type attribute should be set to <b>text</b>, while the type of the value (property <b>ValueType</b>) can be another type, for example, <b>date-time</b> or <b>table</b>.</p>

#### Direction

<b>Element:</b>	Direction
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-

<b>Description:</b>	Type direction (input, output or both) of the property. If not specified, the default value <b>in</b> is used. If a property is specified as an input property, a value can be specified for the property before the action is executed. The value affects the action executed, e.g. results in a different Powershell script. Output properties have no value when the action is executed. Instead, they get a value upon action execution. Possible values are:  In  Out  InOut (also: In-Out, In Out, Out In, Dual, Bidirectional, Both)
---------------------	---

### OutputVariable

<b>Element:</b>	OutputVariable
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	If the <b>Direction</b> element is either <b>Out</b> or <b>InOut</b> , a default output variable name can be specified. Example: %UserTable%.

### Mandatory

<b>Element:</b>	Mandatory
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	A value specifying if the property must be specified for the action. Possible value: <b>Yes</b> or <b>No</b> . If not specified, <b>No</b> is assumed. A value of <b>Yes</b> requires the property value to be specified. If not, action execution will fail. The property is used to check the input of the action before UMRA generates and executes the Powershell script.

### MandatoryProps

<b>Element:</b>	MandatoryProps
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-

<b>Description:</b>	The names (comma separated) of other properties that need to be specified if this property is specified. The property is used to check the input of the action before UMRA generates and executes the Powershell script.
---------------------	--

### MandatoryAbsentProps

<b>Element:</b>	MandatoryAbsentProps
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	The names (comma separated) of other properties that should not be specified if this property is specified. The property is used to check the input of the action before UMRA generates and executes the Powershell script.

### ReturnData

<b>Element:</b>	ReturnData
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No, but needs to be specified if the <b>Direction</b> element is <b>Out</b> or <b>InOut</b>
<b>Attributes:</b>	-
<b>Description:</b>	The outer element of how an output action property should be defined.

### PowershellVariable

<b>Element:</b>	PowershellVariable
<b>Parent element:</b>	ReturnData
<b>Mandatory:</b>	
<b>Attributes:</b>	-
<b>Description:</b>	The name of the powershell variable that will be used in the powershell script (in a script phrase). The contents of this Powershell variable will be transferred to the UMRA variable specified in this ActionProperty.

### DataSpec

<b>Element:</b>	DataSpec
<b>Parent element:</b>	ReturnData
<b>Mandatory:</b>	
<b>Attributes:</b>	dimension
<b>Description:</b>	This data specification element has no content, only the <b>dimension</b> attribute needs a value. The possible values are: <b>value</b> or <b>table</b> . It specifies what kind of data will be returned to this action property.

### Member

<b>Element:</b>	Member
<b>Parent element:</b>	DataSpec
<b>Mandatory:</b>	
<b>Attributes:</b>	type
<b>Description:</b>	The content of this element must be the name of the property in Powershell. Its attribute must specify what type of UMRA variable will get the data.

Series of properties

### PropertiesSeriesSet section

The PropertySeriesSet section specifies what properties of the action must be used in combination with each other. So, when the dynamic action is loaded in UMRA and is used in a Forms, Automation or Mass project, the user fills in the properties that are needed to execute the project successfully. To check whether properties, which are dependent of each other, are filled in correctly, these sets must be specified. With these sets, UMRA checks on run-time whether the user has correctly filled in the right properties. Otherwise, the missing parameters will be mentioned in the log.

For example, there are two ways of specifying a mailbox for the remove-mailbox commandlet. The first option is to identify the mailbox with the -Identity parameter. The second option is to identify the mailbox with the -StoreMailboxIdentity and the -DatabaseName parameter. Obviously, it is not allowed to use, for example, the -Identity parameter in combination with the -StoreMailboxIdentity parameter, vice versa, and also all other combinations with the -DatabaseName parameter. To avoid such conflicts in dynamic actions, the PropertySeriesSet is introduced. Use these tags to specify which properties should be specified together. Check the **Example** section below for a PropertySeriesSet example. The exact name of properties you want to specify as a serie are previously specified in the dynamic action at the **ActionProperties** section in the **Name** element.

#### Example 1:

For the remove-mailbox commandlet action, as mentioned above, the **PropertySeriesSet** should be specified as follows:

```
<PropertySeriesSet>
  <PropertySeries>
    <PropertySerie>Identity</PropertySerie>
    <PropertySerie>Database, StoreMailboxIdentity</PropertySerie>
  </PropertySeries>
</PropertySeriesSet>
```

#### Example 2:

```
<PropertySeriesSet>
  <PropertySeries>
    <DependentPropertyName>User</DependentPropertyName>
    <PropertySerie>OrganizationalUnit, UserName</PropertySerie>
    <PropertySerie>DistinguishedName</PropertySerie>
  </PropertySeries>
</PropertySeriesSet>
```

#### PropertySeriesSet

<b>Element:</b>	PropertySeriesSet
<b>Parent element:</b>	UmraDynamicAction
<b>Mandatory:</b>	No
<b>Attributes:</b>	

<b>Description:</b>	The outer element of all <b>PropertySeries</b> . For each serie of properties that exclude each other, the element contains a child element <b>PropertySeries</b> . Sometimes a dynamic action has multiple series, therefore a <b>PropertySeriesSet</b> can have more than one <b>PropertySeries</b> .
---------------------	---

The **PropertySeriesSet** element contains a child element **PropertySeries**. This element specifies a group of series of which at least one serie must be complete.

### PropertySeries

<b>Element:</b>	PropertySeries
<b>Parent element:</b>	PropertySeriesSet
<b>Mandatory:</b>	Yes, for each PropertySeriesSet
<b>Attributes:</b>	-
<b>Description:</b>	The base element for each serie of properties. This element has mainly 2, sometimes more, child elements. Within the <b>PropertySeries</b> element, UMRA checks its child elements, and at least one child element must have valid (specified) properties. So, in case of the remove-mailbox commandlet, this <b>PropertySeries</b> element has two child elements. One child with the Identity parameter specified in it, and one element with the StoreMailboxIdentity parameter and the DatabaseName parameter in it. UMRA will now check whether the Identity parameter is filled, or if both the StoreMailboxIdentity and DatabaseName parameters are filled. To avoid the possibility of having two or more valid series, specify the <MandatoryAbsentProps> element in the <b>ActionProperties</b> section.

The **PropertySerie** element exists for each **PropertySeries**. It specifies the properties that should be used in combination with each other.

### PropertySerie

<b>Element:</b>	PropertySerie
<b>Parent element:</b>	PropertySeries
<b>Mandatory:</b>	Yes, for each PropertySeries
<b>Attributes:</b>	-



<b>Description:</b>	Use this element to link properties specified in the <b>ActionProperties</b> section with each so they will form a serie. Use commas to specify multiple values. For example: <PropertySerie>StoreMailboxIdentity, DatabaseName</PropertySerie>. The values for this element are previously specified at the <b>ActionProperties</b> section in the <b>Name</b> element.
---------------------	---

### DependentPropertyName

<b>Element:</b>	DependentPropertyName
<b>Parent element:</b>	PropertySeries
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	Use this element to make a serie dependent of a property. The <b>PropertySeries</b> element still has the rule that one serie must be filled in completely, but if a <b>PropertySeries</b> element contains a <b>DependentPropertyName</b> , the series will only be checked if the propertyname in the <b>DependentPropertyName</b> element is specified. So, in case of the second example above, the DistinguishedName must be specified, or the OrganizationalUnit in combination with the UserName. But, in this case, that will only be checked when the User parameter is specified. Specify one property in this element.

### Output specification

The Powershell Agent service offers functionality to return data back to UMRA. Therefore it must send the script to Powershell with the right variables defined in it. The variables defined in the Powershell script will have a connection with the variables in UMRA. Therefore the output variable in UMRA can be filled with the variables in the Powershell script that is sent to the Powershell Agent service. In the XML of the dynamic action you can exactly specify which output variable in UMRA will correspond with which variable you will define in a script phrase.

The following paragraphs will explain how to specify an output variable in XML.

### Single value output data

To return data that consists of a single value, use the following procedure.

1. An action property defines the return data variable:

```
<ActionProperty>
  <Name>CurrentDateTime</Name>
  <Direction>Out</Direction>
```

```
.  
<ReturnData>  
  <PowershellVariable type="value">DateTime</PowershellVariable>  
</ReturnData>  
.  
</ActionProperty>
```

In this example, the element `<Direction>` specifies that the action property is an output property, e.g. when the action is executed by UMRA, an output variable specified for property is filled with the result data. The result data is copied from the Powershell variable `$DateTime`. Note the `$`-sign: the specification does not contain the `$`-character, but in a Powershell script, variables are denoted with the `$`-character. The Powershell script defined in the action should produce the variable `$DateTime`. The value of this variable is copied and returned back to UMRA.

2. The script section defines the variable:

```
<Script>  
  <ScriptPhrase>  
    <Contents>  
      $DateTime=get-date  
    </Contents>  
  </ScriptPhrase>  
</Script>
```

When the script is executed, the Powershell `$DateTime` variable is filled and returned to UMRA. In UMRA, the output variable specified for property `CurrentDateTime` is filled with the result value.

### Table value output data

This topic explains how to return a table generated by the Powershell script back to UMRA. It is more complex to return a table compared to a single value, but the principle is the same: An action property contains the specification of a Powershell variable that is generated by the Powershell script. For a table, the Powershell variable is an array of objects of the same type. Each element of the array results in a row in the table. Different members of each array element make up the columns of the final table.

Example:

```
<ActionProperty>  
  <Name>UserMailboxSimple</Name>
```

```

    .
    <Direction>Out</Direction>
    <OutputVariable>%MailboxTable%</OutputVariable>
    .
    <ReturnData>
        <PowershellVariable
type="object">UserMailbox</PowershellVariable>
        <DataSpec dimension="table">
            <Member type="Text">Name</Member>
            <Member type="Text">SamAccountName</Member>
            <Member type="Text">DistinguishedName</Member>
        </DataSpec>
    </ReturnData>
</ActionProperty>

```

In this example, the UMRA property **UserMailboxSimple** can have an output table variable **%MailboxTable%** specified. The Powershell script generates the variable **\$UserMailbox**. The script section is as follows (simplified):

```

<Script>

    <ScriptPhrase>
        <Contents>
            $UserMailbox=get-mailbox
        </Contents>
    </ScriptPhrase>

</Script>

```

When executed, the Powershell variable **\$UserMailbox** holds an array with mailboxes. Each mailbox is an object of the same type, representing a mailbox. The mailbox objects all have the same members, as defined by Exchange. To setup the output table, the members that must be returned are specified in the property. Each member corresponds with a column of the table. In this example, the members (columns) **Name**, **SamAccountName**, and **DistinguishedName** are returned. To find the possible members, add the Get-Member cmdlet in Powershell: `get-mailbox | get-member`.

The following table illustrate the conversion from a Powershell variable to a UMRA variable:

Powershell array with 2 mailbox objects:

**\$UserMailbox**

--	--

Result table in UMRA with 2 rows:

**%MailboxTable%**


ReturnData element specification

This topic describes the specification of the **ReturnData** element.

**ReturnData**

The ReturnData element is always the child element of an ActionProperty

*Example: The ReturnData and PowershellVariable elements:*

```
<ActionProperty>
  <Name>CurrentDateTime</Name>
  <Direction>Out</Direction>
  .
  <ReturnData ErrorIfNotExist="No">
    <PowershellVariable type="value">DateTime</PowershellVariable>
  </ReturnData>
  .
</ActionProperty>
```

**PowershellVariable**

The element specifies the name of the Powershell variable as generated by the script. The attribute type can have two values:

1. "value": The Powershell variable holds a single value, no DataSpec section must be specified.
2. "object": The Powershell variable holds an array of objects. For the objects, the members to be returned are specified in the DataSpec section. This is the default value, e.g. if the attribute is not specified, the value type="object" is assumed.

*Example: The DataSpec elements:*

```
<ActionProperty>
  <Name>UserMailboxSimple</Name>
  .
  <Direction>Out</Direction>
  <OutputVariable>%MailboxTable%</OutputVariable>
  .
  <ReturnData ErrorIfNotExist="No">
    <PowershellVariable
type="object">UserMailbox</PowershellVariable>
    <DataSpec dimension="table">
      <Member type="Text">Name</Member>
      <Member type="Text">SamAccountName</Member>
      <Member type="Text">DistinguishedName</Member>
    </DataSpec>
  </ReturnData>
</ActionProperty>
```

### DataSpec

The element specifies all the members of the objects that must be returned to UMRA. Each member fills up a column in the result table. The elements supports attribute **dimension**, which can take one of two possible values:

1. "table": The result is an UMRA table. The DataSpec section should contain one ore more **Member** elements.
2. "value": The result is a single UMRA value. The DataSpec section should contain one **Member** element. The Powershell variable should contain a single object.

*Example: The DataSpec element with attribute dimension="value"*

```
<ActionProperty>
  <Name>Owner</Name>
  .
  <Direction>Out</Direction>
  <OutputVariable>%Owner%</OutputVariable>
```

```
.  
<ReturnData ErrorIfNotExist="No">  
  <PowershellVariable  
type="object">SecurityDescriptor</PowershellVariable>  
  <DataSpec dimension="value">  
    <Member type="Text">Owner</Member>  
  </DataSpec>  
</ReturnData>  
</ActionProperty>
```

The example shown above shows the usage of attribute dimension="value" of Powershell variable \$SecurityDescriptor. The corresponding Powershell script contains something like:

```
$SecurityDescriptor=Get-Acl
```

The Powershell variable contains a number of members, including the owner of the file or directory. This value is obtained from the object and returned in UMRA variable %Owner%.

#### **Member**

The element specifies the name of the object member. The element supports the attribute **type** that specifies the UMRA data to be returned. Possible values are: "Text", "String", "Numeric", "Number", "Numerical", "Int", "Integer", "Boolean", "Bool", "Flag", "Date", "Time", "DateTime", "Date-Time". When received by UMRA, the value is converted to the specified type. If not specified, a value of "Text" is assumed.

#### **Attribute ErrorIfNotExist of element ReturnData**

The attribute specifies the error handling. The value of the attribute is either "Yes" or "No". The default value is "No". If not specified, the default value applies. If set to "Yes" the UMRA error handler is activated if no data for the specified UMRA variable is returned from the Powershell Agent service. For a singular value, this is the case if no return data is found for the specified variable. For a tabular value, this is the case if the table is empty, e.g. contains no rows.

*Script section*

The **Script** section specifies the script of the UMRA Powershell dynamic action. In the XML specification file, the script is split up in a number of pieces, called script phrases. Each script phrase, represent a part of the Powershell script. The script element is defined as follows:

<b>Element:</b>	Script
<b>Parent element:</b>	UmraDynamicAction
<b>Mandatory:</b>	No, but normally, an UMRA dynamic Powershell action contains a script. If not, the action doesn't do anything.
<b>Attributes:</b>	
<b>Description:</b>	The outer element of the UMRA Powershell dynamic action script.

The **Script** element contains one or more **ScriptPhrase** elements that define the Powershell script. The order of the child script phrases elements define the order in which they appear in the final Powershell script.

<b>Element:</b>	ScriptPhrase
<b>Parent element:</b>	Script
<b>Mandatory:</b>	Yes, for each scriptphrase, an element should exist.
<b>Attributes:</b>	-
<b>Description:</b>	The scriptphrase contains a piece of the final Powershell script.

Each script phrase, defines a piece of the Powershell script. The script phrases can contain a number of attribute element that define the type of the script phrase and the dependencies on UMRA action properties. Two major script phrase types exist:

1. **Simple script phrase:** A script phrase of which the contents does not depend on the value of any of the action properties. The script phrase only contains a script phrase contents section.
2. **Action property script phrase:** A script phrase of which the content does depend on one or more action properties. The script phrase contains a list with dependent action properties and a script phrase contents section.

The following example shows a script section with one **simple script phrase** and one **action property script phrase**.

```
<Script>

  <!-- a simple script phrase -->
  <ScriptPhrase>
    <Contents>
      Restart-Service Spooler
    </Contents>
  </ScriptPhrase>

  <!-- an Action property script phrase -->
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>

</Script>
```

The script first restarts the **Spooler** service and then restarts the service specified by UMRA action property **ServiceName**.

The following XML specification file shows a full Powershell dynamic action specification, including the script section.

```
<?xml version="1.0" encoding="UTF-16"?>
<UmraDynamicAction version="1" type="Powershell">
  <General>
    <Name>Restart a service</Name>
    <Description>Restart a particular service on the current
computer.</Description>
    <ActionTreeLabels>Powershell, Example actions</ActionTreeLabels>
    <ActionImage>17</ActionImage>
    <Version>101</Version>
```



```

</General>
<ActionProperties>
  <ActionProperty>
    <DisplayName>Service name</DisplayName>
    <Name>ServiceName</Name>
    <Description>The name of the service to restart.</Description>
    <ValueType>Text</ValueType>
  </ActionProperty>
</ActionProperties>
<Script>
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>
</Script>
</UmraDynamicAction>

```

#### Simple script phrase

Simple script phrases only contain a script phrase contents section. The contents of the simple script phrase does not depend on any of the action properties.

<b>Element:</b>	ScriptPhrase
<b>Parent element:</b>	Script
<b>Mandatory:</b>	No
<b>Attributes:</b>	
<b>Description:</b>	The partial Powershell script content. The contents is copied to the final Powershell script.

The following example show a script section with a simple script phrase.

```
<Script>
```

```

    <ScriptPhrase>
      <Contents>
        Restart-Service Spooler
      </Contents>
    </ScriptPhrase>

  </Script>

```

When UMRA executes the action, the following Powershell script is generated:

```
Restart-Service Spooler
```

Action Property script phrase

An **Action Property script phrase** contains a list with dependent action properties and a script phrase contents section. The list with dependent action properties specify how the Powershell script depends on the UMRA properties.

<b>Element:</b>	ScriptPhrase
<b>Parent element:</b>	Script
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	The partial Powershell script content. The contents can contain words (variables) that are replaced by the contents of UMRA property values.

The presence of a single **ActionProperty** child element in the **ScriptPhrase** element makes the script phrase a **Action Property script phrase**.

<b>Element:</b>	ActionProperty
<b>Parent element:</b>	ScriptPhrase
<b>Mandatory:</b>	If the script phrase contents depends on the value of an action property, an <b>ActionProperty</b> element should exist for this action property.
<b>Attributes:</b>	-

<b>Description:</b>	The element defines the dependence between the script phrase contents and the action property.
---------------------	--

The following example shows a script section with an action property script phrase.

```
<Script>

  <!-- an Action property script phrase -->
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>
</Script>
```

At runtime, the value of UMRA property **ServiceName** replaces the word **%ServiceName%** of the **Contents** specification. In this example, the script phrase depends only on a single property: **ServiceName**.

The part of the Powershell script that is generated by an **Action Property script phrase**, can depend on the action properties in several ways:

1. **Conditional inclusion:** The contents of the script phrase is included in the final Powershell script only if certain conditions are met. Example: if some UMRA flag property is set, a cmdlet parameter is added to the Powershell script;
2. **Value dependent:** The contents of the script phrase depends on the value of the action properties. At run-time the value of the specified action properties replace specific parts of the script phrase contents. The value dependence can be straightforward as shown in the previous example, but also more complicated, for instance to repeat a certain Powershell script part to support multi-value UMRA properties.

Any combination of the above dependencies is possible.

The **ActionProperty** always contains the following child element:.

<b>Element:</b>	Name
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	Yes
<b>Attributes:</b>	-
<b>Description:</b>	The name of the action property on which the script phrase depends. The value should correspond with the name of one of the action properties defined earlier in XML specification file.

Next, the ActionProperty element can contain the element *Condition* (see "Conditional Action Property script phrase" on page 49) and/or *Replacement*. See the topics

*Conditional Action Property Script Phrase* (on page 49) and

*Value Dependent Action Property script phrase*

for more information.

Conditional Action Property script phrase

The setup a conditional action property script phrase, add one or more **Condition** elements to the **ActionProperty** element.

<b>Element:</b>	Condition
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No

<b>Attributes:</b>	<b>Criterion</b> The type of condition that determines if the script phrase contents must be included in the Powershell script. Possible values: <ul style="list-style-type: none"> <li>▪ <b>IfNotEmptyOnly:</b> Only include the script phrase contents if the action property is specified and not empty, e.g. contains an non zero value.</li> <li>▪ <b>IfEmptyOnly:</b> Only include the script phrase contents if the action property is not specified or contains an empty or no value.</li> <li>▪ <b>IfTrueOnly:</b> Only include the script phrase contents if the action property value is specified as a boolean true value.</li> <li>▪ <b>IfFalseOnly:</b> Only include the script phrase contents if the action property is specified as a boolean false value.</li> <li>▪ <b>IfSpecifiedOnly:</b> [obsolete]</li> <li>▪ <b>IfNotSpecifiedOnly:</b> [obsolete]</li> </ul>
<b>Description:</b>	The element allows the conditional inclusion of the script phrase contents in the final Powershell script. For instance to include the property for a cmdlet specification only if a certain UMRA property is specified.

The following example shows how to use a conditional action property script phrase. The Powershell cmdlet Restart-Service can also restart dependent service. The parameter -force is used for this purpose. So, to restart a service and all of its dependent service, the Powershell script is:

```
Restart-Service [name of service] -force
```

To start only the service itself, the cmdlet must be specified without the -force parameter:

```
Restart-Service [name of service]
```

To use one and the same action to support both script variants, the UMRA action can include a property, RestartDependentServices, that indicates if the -force parameter must be part of the Powershell script line. In an UMRA project, at runtime, this property can have one of three values:

1. Yes - true: Use the -force parameter;
2. No - false: Do not use the -force parameter;
3. Not specified: Do not use the -force parameter.

So the condition to include the -force parameter is specified by using criterion **IfTrueOnly** in a conditional script phrase.

```
<Script>

  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>

  <ScriptPhrase>
    <ActionProperty>
      <Name>RestartDependentServices</Name>
      <Condition Criterion=IfTrueOnly />
    </ActionProperty>
    <Contents>
      -force
    </Contents>
  </ScriptPhrase>

</Script>
```

#### Value Dependent Action Property script phrase

The contents of a value dependent script phrase depends on the value of related action properties. At run-time the value of the action properties replace specific parts of the script phrase contents. The value dependence can be straightforward, but also more complicated, for instance to repeat a certain Powershell script part to support multi-value UMRA properties.

The following example shows a simple script phrase of which the contents depend on action property **ServiceName**. At run-time, the **%ServiceName%** parameter of the **Contents** element is replaced by the actual value of UMRA action property **ServiceName**.

```
<Script>

  <!-- an Action property script phrase -->
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>
</Script>
```

A value dependent Action Property script phrase always contains a **Replacement** element:

<b>Element:</b>	Replacement
<b>Parent element:</b>	ActionProperty
<b>Mandatory:</b>	No

<b>Attributes:</b>	<p><b>type</b> The type of the replacement determines how the script phrase contents depends on the action property. Possible values are:</p> <p><b>simple:</b> the UMRA property value replaces a part in the script phrase contents. See Simple Value Dependent Action property script phrase for more information.</p> <p><b>multivalue:</b> the UMRA property contains multiple values (0 or more). The script phrase contents must be repeated in the Powershell script, replacing some part of the contents with the current value. See Multi-value Dependent Action Property script phrase for more information.</p> <p><b>boolean:</b> the UMRA action property results in a boolean value, the Powershell script parameter is either <b>\$true</b> or <b>\$false</b>. See Boolean Value Dependent Action Property script phrase for more information.</p> <p><b>[additional type dependent properties, see related topics]</b></p> <p><b>QuoteFormat</b> Always / Default / None (simple and multivalue replacement types only). See QuoteFormat of a Value Dependent Action Property script phrase for more information.</p> <p><b>VariableConversion</b> All/Default/None (simple and multivalue replacement types only). See Variable Conversion of a Value Dependent Action Property script phrase for more information.</p>
<b>Description:</b>	The element contents is the Powershell script contents in which some parts are replaced with UMRA property values.

### Simple Value Dependent Action Property script phrase

In a Simple Value Dependent Action property script phrase, the value of the UMRA property replaces the text in the contents element that is specified in the Replacement element.

In the following example, the value of UMRA property **ServiceName** replaces the text **%ServiceName%** found in the **Contents** element.



```
<Script>

  <!-- an Action property script phrase -->
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement type="simple">%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>
</Script>
```

So if **ServiceName** has a value of **Spooler**, the resulting Powershell script is:

```
Restart-Service Spooler
```

If the **type** attribute is not specified for the **Replacement** element, it is assumed that the action property script phrase is of the **simple** type. Note that the contents of the Replacement element does not need to correspond with the UMRA property name of variable name. The following example produces exactly the same result:

```
<Script>

  <!-- an Action property script phrase -->
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%TheNameOfTheServiceToRestart%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %TheNameOfTheServiceToRestart%
    </Contents>
  </ScriptPhrase>
</Script>
```

## Multi-value Dependent Action Property script phrase

In a Multi-value Dependent Action Property script phrase, the UMRA property is a multi-value property. This UMRA property is either a UMRA text-list or an UMRA table. When UMRA produces the Powershell script, the script phrase contents is repeated for each value of the multi-value property.

The following example shows how to use a multi-value dependent action property script phrase. The script phrase is used to set a number of e-mail addresses for a mailbox. The e-mail addresses originate from the multi-value property EmailAddresses. This property is either an UMRA table or text-list.

For each value, the contents of the **Contents** element is repeated in the resulting Powershell script. In former script phrases, the Powershell variable **\$Mailbox** is set to the mailbox of for instance a user account. The property **EmailAddresses** of the mailbox object contains a list with the current e-mail addresses of the user account. The script phrase first checks if the current e-mail address is contained in the list. If not, it is added.

```
<ScriptPhrase>

  <ActionProperty>
    <Name>EmailAddresses</Name>
    <Replacement type="multivalue">%EmailAddress%</Replacement>
  </ActionProperty>

  <Contents StartWithNewLine="Yes">
    if(!$Mailbox.EmailAddresses.Contains(%EmailAddress%))
  {$Mailbox.EmailAddresses.Add(%EmailAddress%)}
  </Contents>

</ScriptPhrase>
```

In case the multi-value property contains the values smtp:john@tools4ever.co.uk and smtp:john.tools4ever.com, the resulting Powershell script part is as follows:

```
if(!$Mailbox.EmailAddresses.Contains(smtp:john@tools4ever.co.uk))
{$Mailbox.EmailAddresses.Add(smtp:john@tools4ever.co.uk)}

if(!$Mailbox.EmailAddresses.Contains(smtp:john@tools4ever.com)) {
$Mailbox.EmailAddresses.Add(smtp:john@tools4ever.com)}
```

In case the UMRA property is a table, the attribute column is used to refer to the column that contain the target values. If not specified, a default value of 0 is assumed. Example:

```
<Replacement type="multivalue"
column="4">%EmailAddress%</Replacement>
```

#### Boolean Value Dependent Action Property script phrase

The replace text of an Boolean Value Dependent Action Property script phrase is either **\$true** or **\$false**. The **\$true** and **\$false** values are well-known values in Powershell and used to set cmdlet switch parameters. Example: to force a user account to reset the password the next time the user logs on, the **Set-User** cmdlet can be used. For this purpose, the syntax is as follows:

```
Set-User [name of user] -ResetPasswordOnNextLogon $true
```

or

```
Set-User [name of user] -ResetPasswordOnNextLogon $false
```

(Note: This cmdlet uses the Exchange 2007 Powershell snap-in. See *Powershell snap-in* (see "Powershell snap-ins" on page 19) for more information). The corresponding UMRA dynamic Powershell action, contains a properties for the user account (**UserName**) and a flag (**ResetPasswordFlag**) if the switch must be set or reset.

The corresponding script is as follows:

```
<Script>
  <ScriptPhrase>
    <ActionProperty>
      <Name>UserName</Name>
      <Replacement>%UserName%</Replacement>
    </ActionProperty>
```

```
<Contents>
  Set-User %UserName%
</Contents>
</ScriptPhrase>

<ScriptPhrase>
  <ActionProperty>
    <Name>ResetPasswordFlag</Name>
    <Replacement type="boolean">%ResetPasswordSwitch%</Replacement>
  </ActionProperty>
  <Contents>
    -ResetPasswordOnNextLogon %ResetPasswordSwitch%
  </Contents>
</ScriptPhrase>
</Script>
```

#### QuoteFormat of a Value Dependent Action Property script phrase

Powershell requires to enclose text parameters with quotes in certain cases. For instance, when the text parameter contains white space. It is not a good idea, to always enclose text parameter values with quotes: To delete object members, Powershell uses the null-value **\$null** that should not be enclosed in quotes.

The required flexibility is supported by UMRA with the **QuoteFormat** attribute of the **Replacement** element. The **QuoteFormat** can take three values, as shown in the following table:

QuoteFormat mode	Description
Default	Always enclose the replaced text with quotes, except when the value equals \$null. This is the default mode. If the attribute is not specified, this mode is effective.
Always	Always enclose the replaced value with quotes, regardless of the contents of the replace text.

None	Never enclose the replaced value with quotes.
------	---

Powershell supports two types of quotes: single quotes (') and double quotes ("). UMRA always uses double quotes.

Example: When setting e-mail addresses of an Exchange 2007 mailbox, multiple values can be specified with a single comma separated string. The total string should not be enclosed with quotes. In that case, Powershell reads the parameter as a single value. To prevent UMRA from generating a script with quotes, the following script phrase is used:

```
<ScriptPhrase>
  <ActionProperty>
    <Name>EmailAddresses</Name>
    <Condition Criterion="IfNotEmptyOnly" />
    <Replacement QuoteFormat="none">%EmailAddresses%</Replacement>
  </ActionProperty>
  <Contents>
    -EmailAddresses %EmailAddresses%
  </Contents>
</ScriptPhrase>
```

#### Variable Conversion of a Value Dependent Action Property script phrase

With Powershell, cmdlet parameters can contain variable names. For instance, the parameter value

```
User$IdNumber
```

is interpreted as

```
User$IdNumber
```

or as

```
User76893
```

if \$IdNumber is a Powershell variable that equals 76893.

In Powershell, variable names are always preceded by the \$-character. By escaping the \$-character with the ` -character (\$ -> `\$), the subsequent text is not interpreted as a variable name. The required flexibility is supported by UMRA with the **VariableConversion** attribute of the **Replacement** element. The **VariableConversion** can take three values, as shown in the following table:

VariableConversion	Example	Description
Default	abc\$null -> abc\$null -> abc abc\$def -> abc`\$def -> abc\$def	The \$-character is escaped by the ` -character if the name is not <b>null</b> . This is the default behaviour that is also effective when the attribute is not specified.
All	abc\$null -> abc\$null -> abc abc\$def -> abc\$def -> abcDEF	The \$-character is never escaped, e.g. all variables names are interpreted as variable name and as such replaced by Powershell.
None	abc\$null -> abc`\$null -> abc\$null abc\$def -> abc`\$def -> abc\$def	The \$-character is always escaped. All potential variable names are converted to text preceded by the \$-character.

Note: In the table, the variable \$def is supposed to have a value of DEF in Powershell. The examples show sequences of 3 values (value1 -> value2 -> value3):

1. The initial value used by UMRA to be replaced in the script phrase to generate the Powershell script);
2. The updated value by UMRA as stored in the Powershell script;
3. The value as interpreted by Powershell.

#### Script phrase contents

An **Action Property script phrase** always contains a contents element. The contents element specifies the text to be inserted into the Powershell script. The element can contain parts (variable names) that are replaced by property values at run-time.

<b>Element:</b>	Contents
<b>Parent element:</b>	ScriptPhrase
<b>Mandatory:</b>	No, but if not specified, the script phrase does not result in anything.
<b>Attributes:</b>	<p><b>StartWithNewLine</b> Indicates if the text converted to the Powershell script should always start on a new line. Possible value: <b>Yes/No</b>. The default value (not specified) is <b>No</b>.</p> <p><b>StartWithBlank</b> Indicates if the text converted to the Powershell script should always be preceded by a blank or start on a new line. Possible value: <b>Yes/No</b>. The default value (not specified) is <b>Yes</b>.</p> <p><b>Type</b> The resulting Powershell script text can be encrypted. Possible values: <b>Plain / Encrypted</b>. The default value (not specified) is <b>Plain</b>. When <b>Encrypted</b> is specified, the text is encrypted and as such shown in log-files and send to the Powershell service. Note that all data that is exchanged between the UMRA software and the Powershell Agent service is encrypted.</p>
<b>Description:</b>	The partial Powershell script content. The contents can contain words (variables) that are replaced by the contents of UMRA property values.

#### *Session section*

The **Session** section specifies the session maintained by the Powershell Agent service to execute the action.

<b>Element:</b>	Session
<b>Parent element:</b>	UmraDynamicAction
<b>Mandatory:</b>	No, not all dynamic actions require a session. If a dynamic action uses a session, an the session section is not specified, all default values apply.
<b>Attributes:</b>	
<b>Description:</b>	The outer element of the UMRA Powershell dynamic action session specification.

The **Session** element specifies the variables that must be deleted from the Powershell runspace when the action is completed. Even if action execution fails, these variables are removed.

<b>Element:</b>	VariablesCleanup
<b>Parent element:</b>	Session
<b>Mandatory:</b>	No
<b>Attributes:</b>	-
<b>Description:</b>	A comma-separated list with all Powershell variables that are removed when the action is completed. Each variable must be specified without the \$-character. Example: Connection, MailEnvelope

#### *Encrypted properties*

It is possible to encrypt the value of properties, so that the actual value can not be seen in log files, UMRA scripts and so on. For instance, when a dynamic action is used to log on to some system, a password might be required. The value of the password should not be shown in the UMRA projects or log files.

To do so, the UMRA project must use encrypted variables and both the **Properties** and **Script** sections must use some specific settings:

1. In the UMRA project, the data to encrypt must be specified using an **encrypted variable**;
2. In the properties section of the dynamic action specification file, the **ValueType** element must have the attribute **encrypted="Yes"**;
3. In the script section of the dynamic action specification file, the **Contents** element must have the attribute **Type="Encrypted"**.

In such a situation, the data to protect is an UMRA action property that corresponds with a property of the dynamic action. This dynamic action property is used in a script phrase.

#### **UMRA project - encrypted variable**

In the UMRA project, the value to protect must be specified using encrypted variables. This can be accomplished using UMRA action Set encrypted variable or Encrypt text. Both actions produce a variable that holds encrypted text. This variable must be specified for the target property of the dynamic action.

#### **Properties section dynamic action**

Only text properties can be encrypted. For these properties, the **ValueType** specification must include an additional attribute: **encrypted="Yes"**.



```
<ActionProperty>
  <Name>Name</Name>
  <ValueType encrypted="Yes">Text</ValueType>
  <Direction>In</Direction>
</ActionProperty>
```

In UMRA, the property is specified using an encrypted variable. By specifying the encryption attribute, the value is recognized as being encrypted.

### Script section dynamic action

The **Contents** element of the **ScriptPhrase** must contain the attribute **Type="Encrypted"**. This will cause the contents of the script phrase to be encrypted before it is sent to the Powershell Agent service.

```
<ScriptPhrase>
  <ActionProperty>
    <Name>Name</Name>
    <Replacement>%Name%</Replacement>
  </ActionProperty>
  <Contents Type="Encrypted">
    $ACL=Get-ACL %Name%
  </Contents>
</ScriptPhrase>
```

If this attribute is omitted, the actual value is shown in the UMRA log file.

### Dynamic actions library

UMRA maintains an internal library with dynamic actions, including the Powershell dynamic actions. Both the UMRA Console and UMRA Service application maintain a copy of this library. The library contains the complete specifications for all dynamic actions that can be used in UMRA projects. When UMRA loads a project that contains dynamic actions, the actions in the project are updated with the most up-to-date versions of the dynamic action, as defined in the library.

UMRA automatically synchronizes the contents of the libraries so that both the UMRA Console and UMRA Service have an equal contents. When a dynamic action is imported into UMRA, the libraries are updated. Also, when a dynamic action is upgraded, the action is upgraded in the library.

The dynamic action library is located in file **DynActions.dat** in the **Config** directory of both the UMRA Console and UMRA Service application. If the library is deleted and the UMRA application is restarted, it is automatically recreated.

Dynamic actions can only be imported and upgraded with the UMRA Console application. At the UMRA Console application, dynamic actions files must be located in directory **DynamicActions** in the UMRA Console program directory.

#### *Upgrading UMRA dynamic actions*

To upgrade an existing (Powershell) dynamic action, use the following procedure.

1. Increment the version of the dynamic action. The version is found in the XML specification file, element **Version** in the **General** section. See General section of the XML specification file for more information. To version number of a dynamic action that is part of an UMRA script is also shown in the log file. When UMRA executes the dynamic action, a log message is written that shows the version number of the action.
2. Make all changes to the XML specification file.
3. In the UMRA Console application, select the action in the action tree. Right click on the action and select menu option **Reload dynamic action**. The dynamic action is now reloaded and upgraded. Results of the reload operation are shown in the UMRA log. If the UMRA Console is connected to the UMRA Service, the action is also upgraded in the UMRA Service dynamic actions library.

#### *Signature of UMRA dynamic actions*

To prevent accidental changes in dynamic action files, dynamic actions can be protected with a so called signature. The signature is an encrypted value that is calculated from the content of the dynamic action. If the contents of the XML specification file is changed, the signature no longer corresponds with the dynamic action specification. In this case, import and reload operations will fail, e.g. the action is marked as invalid and cannot be imported into UMRA.

To check if a dynamic action contains a signature, check for a signature element in the general section of the XML specification file. If found, the dynamic action is signed.

The signature has the following format:

```
<General>
.
.
.<Signature>Tools4ever:q5GpN2STbiHcQNZINHZoDAs</Signature>
..
..
</General>
```

The name that is part of the signature (Tools4ever) is the owner of the dynamic action file. With a special procedure, the owner can sign an updated version of the XML specification file.

If you want to update an UMRA dynamic action that is signed, use the following procedure:

1. Make a copy of the XML specification file in the UMRA **DynamicActions** subdirectory. Make sure the file contains a name you will recognize later;
2. Give the action a unique name, e.g. change the **Name** element in the **General** section. The new name should be unique. It is recommended to use a file name for the XML specification file that (partially) corresponds with the new name of the action;
3. Remove the **Signature** element from the **General** section in the new XML specification file;
4. Save the XML specification file;
5. *Import* (see "UMRA dynamic action example: Import the dynamic action" on page 70) the new XML file with the UMRA Console application. The new action contains no signature and can be updated and modified later.

#### *Remove a dynamic action*

Once a new dynamic action is added, it is stored in the dynamic actions library. A dynamic action is not removed by removing the xml specification file. Instead, use the following procedure:

1. Stop the UMRA Service.
2. Close UMRA Console application.
3. Go to the location where the UMRA Console application is installed. By default this is "C:\Program Files\Tools4ever\User Management Resource Administrator".
4. From the folder **DynamicActions**, remove the dynamic action specification file. This is an xml file with extension **.xml**.
5. From the folder **Config**, remove the file **DynActions.dat**. (The dynamic action library will rebuild automatically when the UMRA Console application is restarted.)
6. Go the UMRA Service directory. By default, this is "C:\Program Files\UmraService" on the server on which the UMRA Service is installed.
7. From the folder **Config**, remove the file **DynActions.dat**. (The dynamic action library will rebuild automatically when the UMRA Service application is restarted.)
8. Start the UMRA Service.
9. Start the UMRA Console application.

Note: UMRA projects that contain dynamic actions that are removed will remain intact. The actions are stored in the script of the project and will execute as before.

### Example

This chapter describes a complete example to create and use a new UMRA dynamic action. The dynamic action performs a simple task with Powershell, e.g. restarts a particular service on the current computer. The example shows how to create a new UMRA dynamic action to execute the requested Powershell script.

To restart the printer spooler service on the current computer, the following Powershell script can be used:

```
Restart-Service Spooler
```

The commandlet **Restart-Service** is part of the default set of Powershell commandlets. The Exchange 2007 management tools don't need to be installed to use this commandlet. The name of service to be restarted is **Spooler**. This is the print spooler service. Note: the name **Print Spooler** is the display name. The name **Spooler** is the (real) service name. Use the name of another service to restart that particular service.

The corresponding UMRA action of this Powershell script is an action to restart a particular service. The name of the action is for instance **Restart service**. One of the properties of the action is the name of the service.

### *UMRA dynamic action example: Goal*

The goal of this example is to add an dynamic action to UMRA to restart a particular service on the current computer. The action is always the same: restart the service. The name of the service to restart is a parameter of the action.

This is an important aspect: it determines the behaviour of the UMRA action. The action always restarts the service. But it does not always restart the same service. The service is a parameter of the action. With UMRA, any Powershell script can be executed. The exact behaviour of the UMRA action is up to the designer of the action: all options can be implemented. In this particular case, for instance the following UMRA dynamic actions are possible:

UMRA Action	Parameters	Description
-------------	------------	-------------

Restart service	Name of service	Restart the particular service
Restart spooler service	[none]	Restart the spooler service. The action does not allow to restart another service.
Manage service	Action, Name of service	Start, restart or stop a particular service. Both the type of the action and the target service are parameters of the UMRA dynamic action.

It is up to the designer of the UMRA dynamic action to determine the functionality of the action and the parameters of the action. In this example, we will design the action to always restart a service. The name of the service is a parameter of the action. As an exercise, you can later implement the other dynamic actions.

*UMRA dynamic action example: XML file*

The UMRA dynamic action to restart a particular service on the local computer is defined in an XML-file:

```
<?xml version="1.0" encoding="UTF-16"?>
<UmraDynamicAction version="1" type="Powershell">
  <General>
    <Name>Restart a service</Name>
    <Description>Restart a particular service on the current
computer.</Description>
    <ActionTreeLabels>Powershell, Example actions</ActionTreeLabels>
    <ActionImage>17</ActionImage>
    <Version>101</Version>
  </General>
  <ActionProperties>
    <ActionProperty>
      <DisplayName>Service name</DisplayName>
      <Name>ServiceName</Name>
      <Description>The name of the service to restart.</Description>
      <ValueType>Text</ValueType>
    </ActionProperty>
  </ActionProperties>
  <Script>
    <ScriptPhrase>
      <ActionProperty>
        <Name>ServiceName</Name>
        <Replacement>%ServiceName%</Replacement>
      </ActionProperty>
    </ScriptPhrase>
  </Script>
</UmraDynamicAction>
```

```
        </ActionProperty>
        <Contents>
            Restart-Service %ServiceName%
        </Contents>
    </ScriptPhrase>
</Script>
</UmraDynamicAction>
```

The XML-file completely defines the UMRA action and the Powershell script. The Powershell Agent Service will execute the Powershell script when the UMRA software executes a project that contains the action: When UMRA executes the action, the Powershell script is composed and send to the Powershell Agent Service with the request to execute the script.

The XML-file contains four main parts:

1. **General section:** The section defines the name of the UMRA action, the description and the position in the action tree and version information.
2. **Configuration section:** A number of configuration settings;
3. **ActionProperties section:** All the UMRA properties of the action. The properties show up in UMRA when the action is configured.
4. **PropertiesSeriesSet:** The relationships between
5. **Script section:** The specification of the Powershell script. When UMRA executes the action as part of a project, the script is composed according to this specification and sent to the Powershell Agent Service.

The next topics deals with the different sections. For the example, no special configuration settings apply. Hence, the configuration section is not specified for the example.

*UMRA dynamic action example: XML file - general section*

The general section of the UMRA dynamic action file defines the general settings of the UMRA action.

The XML file start with the declaration of the file:

```
<?xml version="1.0" encoding="UTF-16"?>
```

Next, the action element is opened. In this case, the dynamic action type is Powershell, e.g. the dynamic action is a Powershell action. The complete action specification is contained the element **UmraDynamicAction**.

```
<UmraDynamicAction version="1" type="Powershell">
```

The **General** element specifies the name and description of the service. The name of the action is the name as shown in the UMRA action tree and UMRA projects.

```
<General>
  <Name>Restart a service</Name>
  <Description>Restart a particular service on the current
computer.</Description>
  <ActionTreeLabels>Powershell, Example actions</ActionTreeLabels>
  <ActionImage>17</ActionImage>
  <Version>101</Version>
</General>
```

The **ActionTreeLabels** element defines the position in tree. The element contents is a comma separated list. Each entry corresponds with a folder of the UMRA action tree. If the folder does not exist, it is created automatically when the action is loaded.

The **ActionImage** element specified the image shown for the action. The **Version** element is used to upgrade the action if the specification in the XML-file is updated.

*UMRA dynamic action example: XML file - ActionProperties section*

The **ActionProperties** section of the XML-file specifies all of the properties of the UMRA action.

```
<ActionProperties>
  <ActionProperty>
    <DisplayName>Service name</DisplayName>
    <Name>ServiceName</Name>
    <Description>The name of the service to restart.</Description>
    <ValueType>Text</ValueType>
  </ActionProperty>
</ActionProperties>
```

All properties are contained in the element **ActionProperties**. For each property, a child element **ActionProperty** exists. The **ActionProperty** element, supports a number of child element. In this

example, only the most important child elements are specified. For all other possible element, the default values apply:

1. **DisplayName:** The display name of the property. This is the name shown in UMRA. It is a user-friendly name and it can contain spaces etc.
2. **Name:** The name of the property. This name is used in other sections of this XML-file. The name is used to refer to this property.
3. **Description:** The description of the property.
4. **ValueType:** The UMRA value type of this property. Most properties have **text** values.

When an UMRA project contains the dynamic action, the properties list of UMRA shows the properties that are specified in this section. To add another property, a new element **ActionProperty** must be added as a child element to the element **ActionProperties**. In this example, the action only has a single property.

*UMRA dynamic action example: XML file - Script section*

The script section of the XML-file specifies the Powershell script. The specification includes the dependencies of the script and the UMRA action properties.

```
<Script>
  <ScriptPhrase>
    <ActionProperty>
      <Name>ServiceName</Name>
      <Replacement>%ServiceName%</Replacement>
    </ActionProperty>
    <Contents>
      Restart-Service %ServiceName%
    </Contents>
  </ScriptPhrase>
</Script>
```

The script is made up from a number of **ScriptPhrases**, each with its own specification. Each script phrase results in a part of the final Powershell script that is sent to the Powershell Agent Service. The contents of single script phrase strongly can depend on the content of UMRA property values. The script of the example only contains a single script phrase. This script phrase depends on a single property: **ServiceName**. This name refers to the UMRA action property specified in the **ActionProperties** section. The **ActionProperty** specifies a replacement-type script phrase: At runtime, the actual value of UMRA action property **ServiceName** replaces the occurrence of **%ServiceName%** in the script phase contents specification.



The contents of the script phrase is

```
Restart-Service %ServiceName%
```

So at runtime, the actual value of property **ServiceName** replaces the word **%ServiceName%**. In case the value of the property equals Spooler, the final Powershell script equals:

```
Restart-Service Spooler
```

This script restarts the spooler service of the local computer.

*UMRA dynamic action example: Import the dynamic action*

To import the action in UMRA, create the XML file and store the file to the UMRA program subdirectory DynamicActions. By default, this is directory:

C:\Program Files\Tools4ever\User Management Resource Administrator\DynamicActions

Note that the file must have a unique name and that the name of the action, as specified in the general section must be unique. The name of the action identifies the action in UMRA.

Start the UMRA Console application and select option **Tools, Import dynamic action**. This brings up a dialog showing the UMRA dynamic action files that are not already imported into to UMRA dynamic action library.

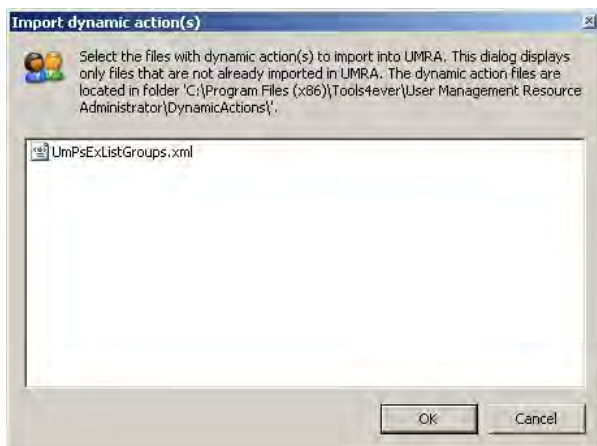


Figure: Import dynamic action(s)

When you select the file and click OK, the file is imported into the UMRA library with dynamic actions. The log reflects the status of the import procedure. (When the UMRA Console is started, and the file is

already located in the directory, it is imported automatically.) When the action is imported, it is added to the tree with action items as shown in the following figure.

[screenshot]

The new action can now be used in UMRA projects.

If there are no new dynamic actions available in the DynamicActions folder the following message will appear:

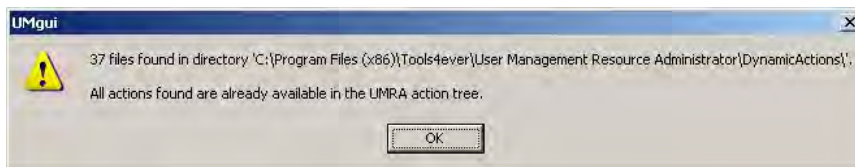


Figure: All dynamic actions are already available in the UMRA action tree

#### *UMRA dynamic action example: Using the dynamic action*

In this example, the new UMRA dynamic action is used in a simple form project. To do so, connect to the UMRA Service and start and setup a new form project. The form only needs a single button that initiates the execution of the project script. Don't forget to setup the project security. Note that when the connection with the UMRA Service is setup, the UMRA dynamic action libraries maintained by the UMRA Console and UMRA Service application are synchronized. Add the new action to the script of the project.

[screenshot of project]

In this example, the action **Restart a service**, only has a single property: **Service name**. You can specify any valid service name for the value property. To restart the **Print Spooler** service, specify **Spooler**.

[screenshot with properties]

Before you run the project, confirm that the UMRA Service is *connected to the Powershell Agent Service* (see "Powershell Agent connection settings" on page 5). If this is not the case, project execution will fail. Now run the script in the UMRA Console preview window. The UMRA Service will execute the action. To execute a Powershell action, UMRA performs the following steps:

1. Create the Powershell script as determined by the action specification and UMRA property values;
2. Send the Powershell script to the Powershell Agent Service;
3. Process any returned results.

The procedure results in the following log:

```
Starting User Management Resource Administrator log session, build
1419 at 08:26:16 01/23/2008
08:26:16 01/23/2008 Submit form to UMRA service on computer 'ZEUS'.
08:26:16 01/23/2008 Executing form submit procedure, UMRA build 1419,
account: 'TOOLS4EVER162\J. Vriens', form project: 'Form' ('31-
72861d28-e496-4492-8a18-d15faf6d29ea').
08:26:16 01/23/2008 Variable 1:
%UmraFormSubmitAccount%=TOOLS4EVER162\J. Vriens
08:26:16 01/23/2008 Variable 2: %UmraClientComputerName%=ZEUS
08:26:16 01/23/2008 Variable 3: %NowDay%=23
08:26:16 01/23/2008 Variable 4: %NowMonth%=01
08:26:16 01/23/2008 Variable 5: %NowYear%=2008
08:26:16 01/23/2008 Variable 6: %NowHour%=08
08:26:16 01/23/2008 Variable 7: %NowMinute%=26
08:26:16 01/23/2008 Variable 8: %NowSecond%=16
08:26:16 01/23/2008 Execution UMRA Powershell action 'Restart a
service', version '101' ...
08:26:16 01/23/2008 Line 01: Restart-Service "Spooler"
08:26:19 01/23/2008 T4ePowerShell session 1002: Successfully created
Windows PowerShell execution environment.
08:26:19 01/23/2008 T4ePowerShell session 1002: Setting up and
executing PowerShell pipeline commands...
08:26:21 01/23/2008 T4ePowerShell session 1002: W9: Waiting for
service 'Print Spooler (Spooler)' to finish starting...
08:26:22 01/23/2008 T4ePowerShell session 1002: Pipeline invocation
result object 0
08:26:22 01/23/2008 T4ePowerShell session 1002: Finished setting up
and executing PowerShell pipeline commands.
08:26:22 01/23/2008 Form message:
'01/23/2008,08:26:16,"TOOLS4EVER162\J. Vriens","Form submit",OK,Form'
End of session
```

The log includes the Powershell script (Restart-Service "Spooler") and all logging information produced by the Powershell Agent service (all lines starting with T4ePowerShell session...).

### 3.13.8. Manage Active Directory with the UMRA Powershell Agent service

A number of UMRA dynamic actions require the **Exchange 2007 Management Tools** to be installed, although these actions do not manage Exchange 2007 related resources. For these actions, it is not

necessary to have Exchange 2007 Server installed on any server, but the Exchange 2007 Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service. Examples of these UMRA actions are: **Get AD permissions** (action folder: Powershell, Active Directory permissions) and **Get (nested) group memberships** (action folder: Powershell, Group management). These actions are implemented using cmdlets that are part of the Exchange 2007 *snap-in*

**Microsoft.Exchange.Management.PowerShell.Admin**. Therefore, the Exchange Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service in order to use these actions. Note that not all cmdlets of this snap-in can be used if Exchange 2007 server is not installed. For instance, the cmdlets to create a mail-enable user account requires Exchange 2007 Server to be installed.

The described UMRA dynamic actions have the following characteristics:

1. The actions use cmdlets that are part of the Exchange 2007 Powershell snap-in that comes with the Exchange 2007 Management Tools;
2. The cmdlets do not require Exchange 2007 Server to be installed in the network. Instead, the cmdlets can be used to manage Active Directory resources;
3. To execute the Powershell scripts that use these cmdlets, the Exchange 2007 Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service.

The Exchange 2007 Management Tools are available for 32-bit and 64-bit platforms. When Exchange 2007 Server is installed (64-bit platform only), the tools are installed automatically. To install the tools on a 32-bit platform, see *Setting up the Exchange 2007 Management Tools on a 32-bit platform* for more information. To install the tools on a 64-bit platform without installing Exchange 2007 Server, a similar procedure must be used.

### 3.13.9. Managing Exchange 2003 with the UMRA Powershell Agent service

With the UMRA Powershell Agent service, it is possible to manage a number of Exchange 2003 mailbox settings, including Exchange 2003 mailbox permissions. Special configuration settings apply to support this type of functionality.

#### Environment

This section describes the principle of the required environment to support the functions to manage Exchange 2003 with UMRA and the Powershell Agent service. The environment runs Active Directory on Windows 2003, and one or more Exchange 2003 servers. Note that there is no server running Exchange 2007. The following systems are part of the environment:

1. **Domain controller:** There must be at least a one domain controller to run Active Directory. The domain controller must run Windows 2003 in native mode.

2. **Exchange 2003:** One or more servers run Exchange 2003 Server.
3. **UMRA:** The UMRA software (Console and Service) can be installed on any computer, except for the Powershell Agent service.
4. **UMRA Powershell Agent service:** The Powershell Agent service is installed by using the UMRA Console application. The service cannot be installed on a computer that runs Exchange 2003 server but it can be installed on any other server that is part of the domain. On this computer, the **Exchange 2007 Management Tools** must be installed. For a procedure to install the tools on a 32-bit platform, see *Setting up the Exchange 2007 Management Tools on a 32-bit platform*.

Not all cmdlets can be used to manage Exchange 2003 mailboxes. Valid cmdlets are: Get-Mailbox, Get-User, Add-MailboxPermission, Add-AdPermission, Set-Group. As a general rule, the cmdlets that access Active Directory instead of the the Exchange 2007 server can be used to manage Exchange 2003 mailboxes.

### **3.13.10. Setting up the Exchange 2007 Management Tools on a 32-bit platform**

#### **How to setup the Exchange 2007 Management Tools on a 32-bit platform**

The **Exchange 2007 Management Tools** are available both for 32-bit and 64-bit platforms. **Exchange 2007 Server** is available only for 64-bit platforms, but the **Exchange 2007 Management Tools** run on both 32-bit and 64-bit platforms. This topic describes how to setup the 32-bit **Exchange 2007 Management Tools** on a 32-bit Windows 2003, Service Pack 2 platform. Note that the **Exchange 2007 Management Tools** can also be installed on Windows XP. For this platform, the procedure is similar.

1. Log on to the computer with domain administrative access. Make sure the domain administrator is an administrator of the local computer as well.

2. If not installed, configure the **Microsoft Internet Information Services Common Files** using **Control Panel, Add or Remove Programs**. Select item **Application Server**, click **details**, select **Internet Information Services (IIS)**, click **details** and check item **Common Files**. (If the item is already checked, the **Common Files** are already installed.) Click **OK** and **Next** a number of times to confirm the selection and start the installation.

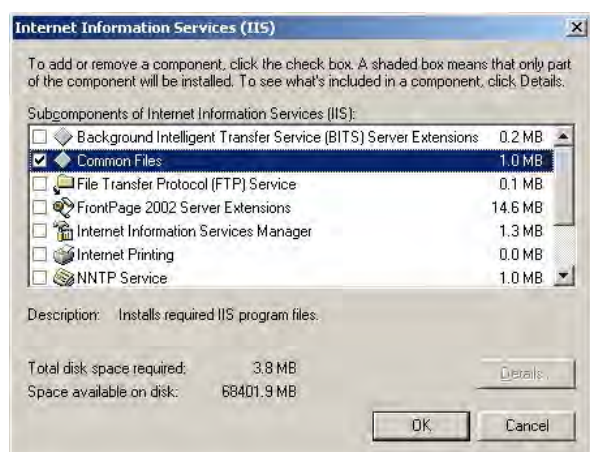


Figure: IIS common files.

3. Verify the **domain functional level**. The level must be **Windows 2000 native mode** or above. To raise the level, run **Active Directory Users and Computer** on a domain controller, right click the domain and select option **Raise Domain Functional Level**.
4. Exchange 2003 only: Verify the **operation** mode of the Exchange Organization. In the **Exchange System Manager**, right click on the organization and select menu option **Properties**. Check the contents of the **Operation Mode** field.
5. Install the pre-requirements component: **Microsoft .NET Framework Version 2.0**. To download, visit link <http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en>
6. Install the pre-requirements component: **Microsoft .NET Framework Version 2.0** hotfix. To download, visit link <http://www.microsoft.com/technet/prodtechnol/exchange/Analyzer/729d1648-ff17-43f9-a1cf-4285a82d4917.mspx?mfr=true>
7. Install the pre-requirements component: **Microsoft Management Console (MMC) 3.0**. To download, visit link <http://www.microsoft.com/downloads/details.aspx?familyid=4C84F80B-908D-4B5D-8AA8-27B962566D9F&displaylang=en>
8. Install the pre-requirements component: **Windows PowerShell**. To download, visit link <http://www.microsoft.com/downloads/details.aspx?familyid=10EE29AF-7C3A-4057-8367-C9C1DAB6E2BF&displaylang=en>
9. Download the 32-bit Microsoft Exchange 2007 installation files from: <http://www.microsoft.com/downloads/details.aspx?FamilyId=444C259E-605F-4A82-96D5-A2F448C9D4FF&displaylang=en>
10. Extract the files and start the setup.

## 11. Selection option **Install Microsoft Exchange**.

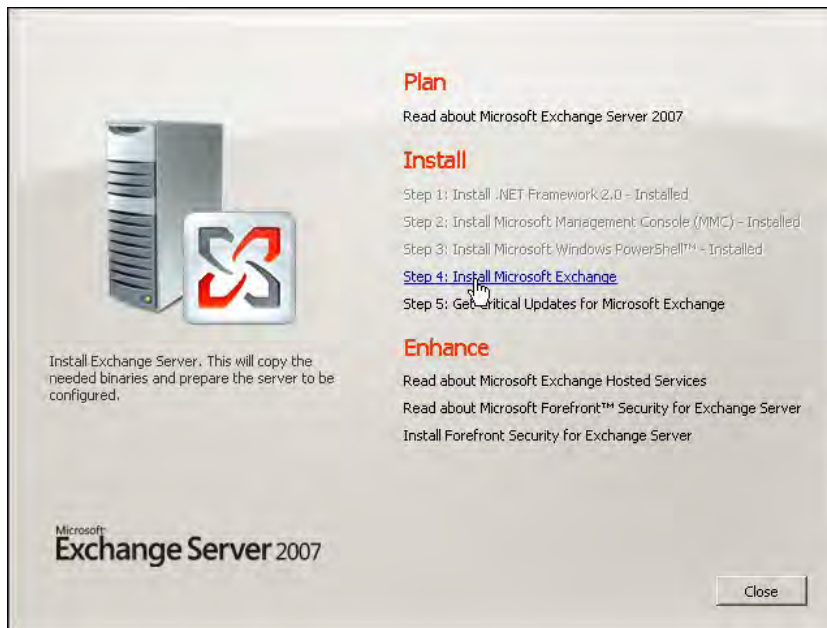


Figure: Exchange Server 2007 Init

## 12. Proceed with the wizard and when asked, selection the option **Custom Exchange Server Installation**.

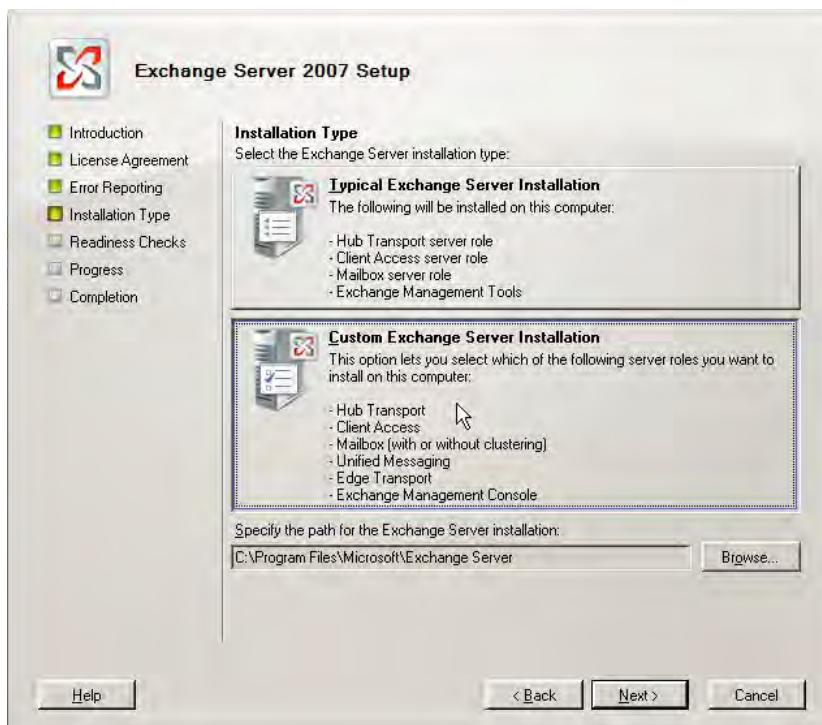


Figure: Exchange Server 2007 Installation type



13. To install only the **Management Tools**, select the appropriate option.

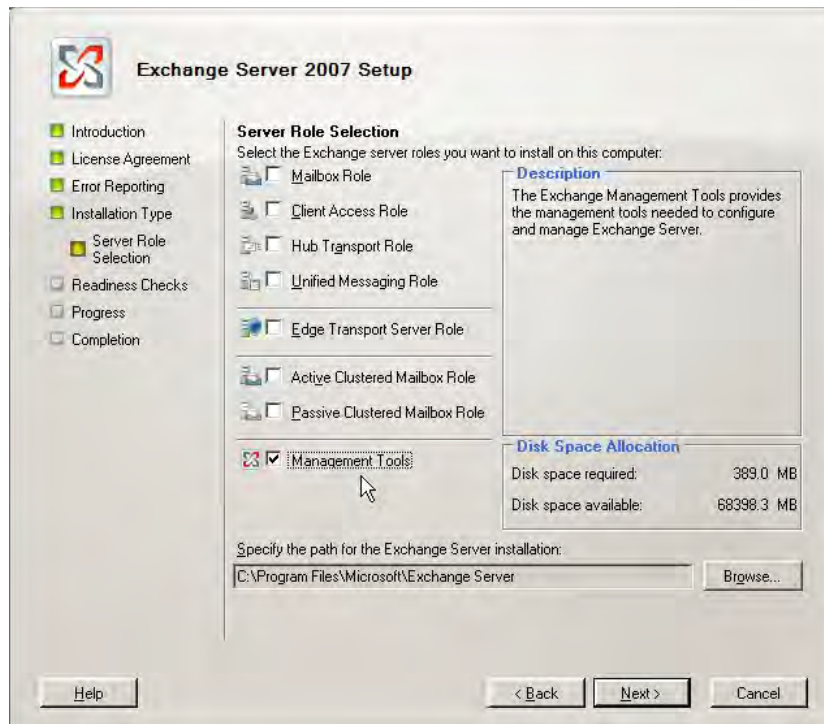


Figure: Exchange Server 2007 server role selection.

14. First, some tests are performed. When ready, select option **Install**.

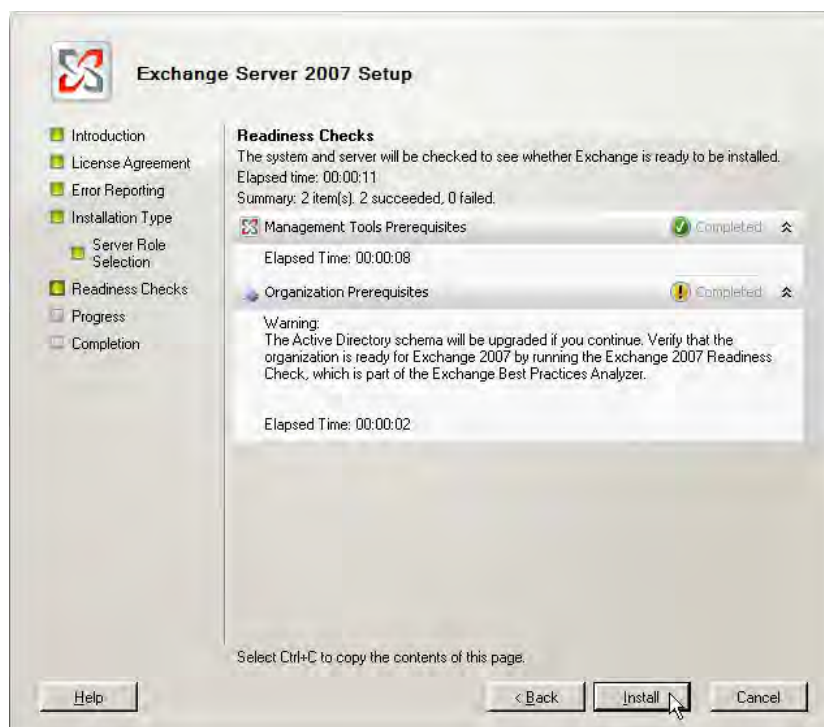


Figure: Exchange Server 2007 readiness checks



15. The **Exchange 2007 Management Tools** are now installed.

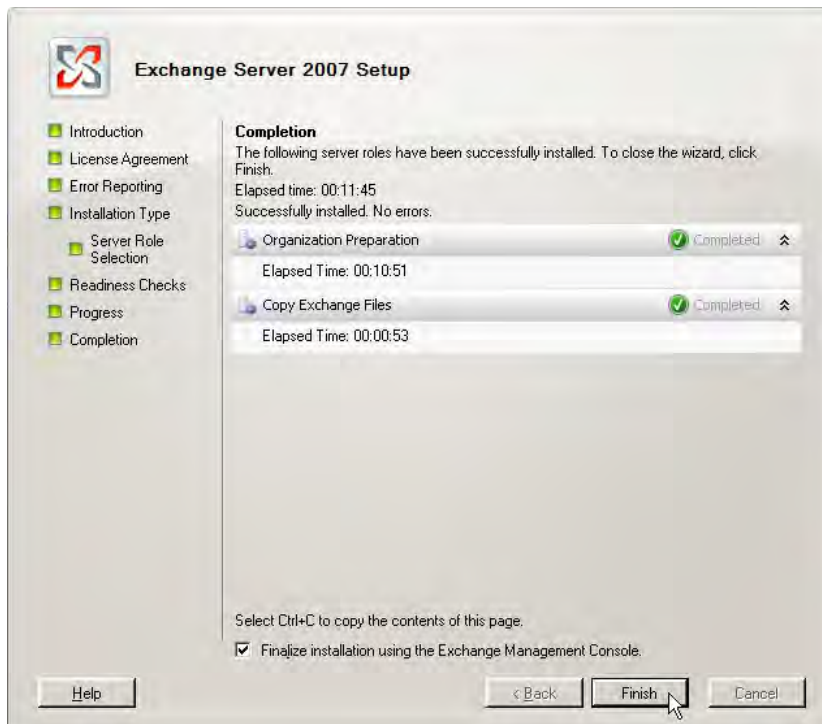


Figure: Exchange Server 2007 completion

16. When read, click **Finish** and exit the application. To test the installation, select program **All Programs, Microsoft Exchange Server 2007, Exchange Management Shell** or **Exchange Management Console**.

### 3.13.11. Powershell Agent service session

A Powershell script is always executed in a runspace. The runspace provides the communications between the Windows PowerShell runtime and the Powershell Agent service host application. The runspace also holds a list with Powershell variables. The Powershell Agent service is able to create and terminate runspace. When a runspace is terminated, all runspace variables are lost. Sometimes it is convenient to run consecutive Powershell scripts using the same runspace. Specially when variables created in one Powershell script are used by the next script.

Example: In a SOAP environment, the SOAP methods supported by a web-service can be made available through a so call proxy object. The proxy object can be created using a Powershell script and stored in a Powershell runspace variable. A Powershell script can then use the methods of the proxy to execute SOAP calls. It is very inefficient to re-create the proxy each time a SOAP call is made. It is more efficient is to create the proxy object once and use it in all subsequent scripts. In this case, all scripts must be

executed in the same runspace. The first script creates the proxy object variable and consecutive scripts running in the same runspace use the variable.

The Powershell Agent service can execute scripts in 2 ways:

1. **Script execution without a session:** For each script, a runspace is created. The script is executed in the context of the runspace and finally the the runspace is deleted. This is the most simple approach. Powershell variables only exist as long as the script is executed.
2. **Script execution with a session:** First, a session is created. The session contains the runspace and an identifier (Session ID) that is used to refer to the session. Next, a number of Powershell scripts can be executed using the the same runspace. Each script refers to the session to identify the correct session. Finally, the session is released, thereby removing the Powershell runspace. In this scenario, additional actions are used to setup and release the Powershell session.

With UMRA, dynamic actions are used to execute Powershell scripts. The configuration section of the dynamic action specifies if an existing Powershell Agent service session must be used. In this case, the dynamic action has a property **Session ID** that refers to the session.

The following section describes the principle of operation of both methods:

**Script execution without a session:**

1. The dynamic action specified that no session is used. This is the default specification.
2. When the action is executed, the Powershell Agent service creates a runspace that exists for the duration of the script. When the script is completed, the runspace is destroyed, including all Powershell variables that are created and used during script execution.

**Script execution with a session:**

1. A Powershell Agent service session is created with UMRA action **Setup Powershell Agent service session** (action folder: **Powershell, Agent service session**). The action returns a reference value, **Session ID**, to be used in subsequent dynamic actions that use the session.
2. A Powershell script that uses the session is executed. The configuration section of the dynamic action specifies that a session is used:

```
...  
<Configuration>  
  <Session required="Yes"/>  
</Configuration>  
...
```

The action contains an mandatory property, **Session ID**, that refers to the session:

```
...
```

```

<ActionProperty>
  <DisplayName>Session ID</DisplayName>
  <Name>SessionID</Name>
  <Description>This parameter identifies the Powershell Agent
service session.</Description>
  <ValueType>Numeric</ValueType>
  <DefaultValue
type="text">%PowershellAgentSessionId%</DefaultValue>
  <Direction>in</Direction>
  <Mandatory>Yes</Mandatory>
</ActionProperty>
. . .

```

In UMRA, the session ID is stored in variable %PowershellAgentSessionId%.

3. Step 2 is repeated a number of times, all actions using the existing Powershell Agent service session. (This is not mandatory: another Powershell action can be executed that uses its own session).
4. The Powershell Agent service session is released with UMRA action **Release Powershell Agent service session** (action folder: **Powershell, Agent service session**). The input of the action is the **Session ID** that refers to the existing session. When the Powershell Agent service session is idle for a long period, the session is released automatically. The default maximum idle interval time is 4 hours (240 minutes). To configure this time, see Registry settings, key **SessionTimeToLive**, for more information.

For more information on the Powershell Agent service session, see the following topics:

*Powershell Agent service session on page 78*

*Script Action: Setup Powershell Agent service session*

*Script Action: Release Powershell Agent service session on page 606*

*Configuration section on page 27*

---

## CHAPTER 1

### 3.13.12. UMRA Sessions

UMRA Sessions are used to store variables, even when projects are finished. Further, the session is used to identify the UMRA client. In UMRA, a project maintains a variable list. The main parent project can access and execute other child projects, passing the variable list. Child projects are initiated with UMRA actions like the **For-Each** and **Execute script** action. When the parent project terminates, the variable list is destroyed.

In certain circumstances it is efficient to share variables between distinct UMRA (parent) project executions, e.g. to store one or more variables, even when a parent project is completed. This is for instance true in the following situation:

1. Projects are initiated from the UMRA Forms client or the UMRA COM object (using ASP or ASPX);
2. The projects use the Powershell Agent service to execute Powershell actions that use a Powershell Agent service session.

The necessity to store UMRA variables in this environment is described below, using an example. In this example, a number of UMRA Forms are used to manage an environment that is accessed using SOAP. The SOAP proxy object is created and stored at the Powershell Agent service. Each form initiates some SOAP oriented transaction. The SOAP call is made using a SOAP proxy object that exposes the methods that are supported by the web-service at the remote site. The SOAP object is maintained by the Powershell Agent service and stored as a Powershell variable in a *Powershell Agent service session* on page 78. So between the UMRA Service and the Powershell Agent service, a session exists. This allows the UMRA Service to execute Powershell scripts that use objects (e.g. the SOAP proxy) that exist in the Powershell runspace, part of the session.

If now different forms are submitted, each form will initiate a new UMRA project, that starts with a new variable list. In order to be able to use an existing Powershell session, the variable that is used to refer to the session should be stored beyond the UMRA project boundary. If this is not the case, it is not possible to use an existing Powershell Agent service session in different UMRA parent projects. To support this mechanism UMRA supports sessions. The UMRA session is maintained at the UMRA Service (UMRA Console for mass projects). Depending on

the type of UMRA project execution, a session remains active even if a parent project is terminated. A session contains a variable list, that can store variables, for instance the session id of a Powershell Agent service session.

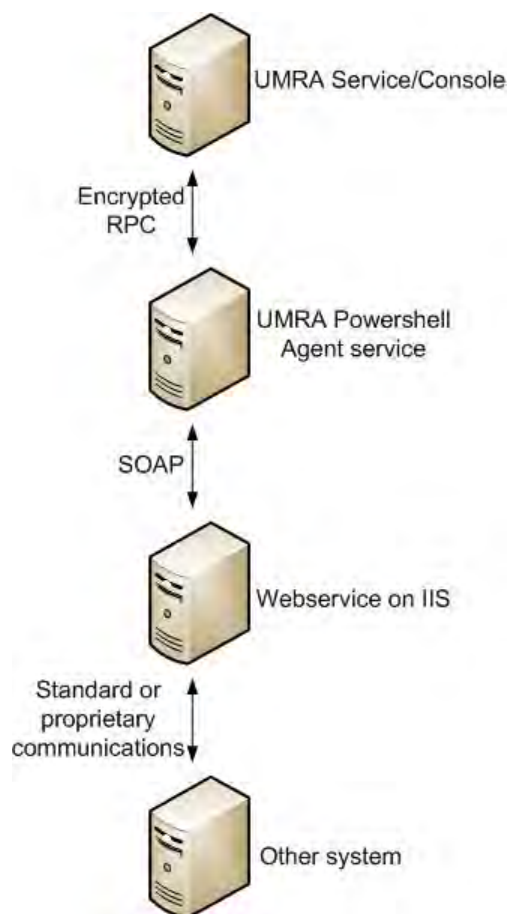
UMRA project execution	Session starts when...	Session ends when...	Session control
<b>UMRA Forms</b>	The UMRA Forms client starts. As long as the application runs, the same session remains active. Different forms share the same session. If two instances of the UMRA Forms application are started on the same computer, two sessions exist.	The UMRA Forms client terminates.	Transparent, automatic.
<b>UMRA COM object (ASP,ASPX)</b>	The UMRA COM object is created.	Programmatic control.	Using COM methods: <i>GetConnectionString</i> , <i>RestoreConnection</i> , <i>ReleaseConnection</i> on page 6
<b>UMRA Command line interface</b>	The UMRA Command line interface starts.	The UMRA Command line interface terminates.	Transparent, automatic
<b>UMRA Mass</b>	The UMRA Console application starts.	The UMRA Console application terminates.	Transparent, automatic
<b>UMRA Scheduled automation project</b>	Just before the parent project is started.	The parent project is completed.	Transparent, automatic.

Each UMRA session contains a variable list. Only persistent variables can be stored in the list (e.g. text, boolean, numerical and table values). UMRA supports several actions to manage the variables in the session variable list: *Script Action: Get session variable* on page 566, *Script Action: Set session variable*, *Script Action: Check session variable*, *Script Action: Delete session variable*.

Note: When a session maintained by the UMRA Service is not used for more than 24 hours, the service will automatically release the session.

### **3.13.13. Configuring a secure web-site with IIS**

The UMRA Powershell Agent service is often used to implement a connector with a system that supports SOAP. In such a scenario, the UMRA Console or UMRA Service application execute scripts that contain dynamic actions that are executed by the UMRA Powershell Agent service. The agent service communicates with a webservice, often running on top of IIS. This topic describes how to setup the webservice in a secure manner, and how to obtain and configure certificates in this scenario. The image below shows the different systems. Note that in practice, all software components can also run on one and the same computer.



In this procedure example, it is assumed that the operating system of the computer that hosts the webservice is Windows Server 2003. For different operating systems, like Windows XP, Windows Vista, or Windows Server 2008, a similar procedure applies.

#### **Step 1: Obtain a certificate**

To setup the secure environment, a valid certificate is required. From this certificate, you need the private key and the certificate as well. You can obtain such a certificate from a certification authority like VeriSign, or ask Tools4ever for such a certificate. To generate the certificate, you need to provide the certification authority with specific information:

1. **Name:** The DNS name of the computer that runs IIS. Example: saturn.tools4ever.com. Note that this name must be correct, if not, the communications that rely on the certificate will fail;
2. **E-mail:** The email address of the person that should be contacted for the certificate information. Example: jsmith@tools4ever.com
3. **Company:** The name of the organization requesting the certificate;
4. **Department**
5. **City**
6. **State**
7. **Country/Region**

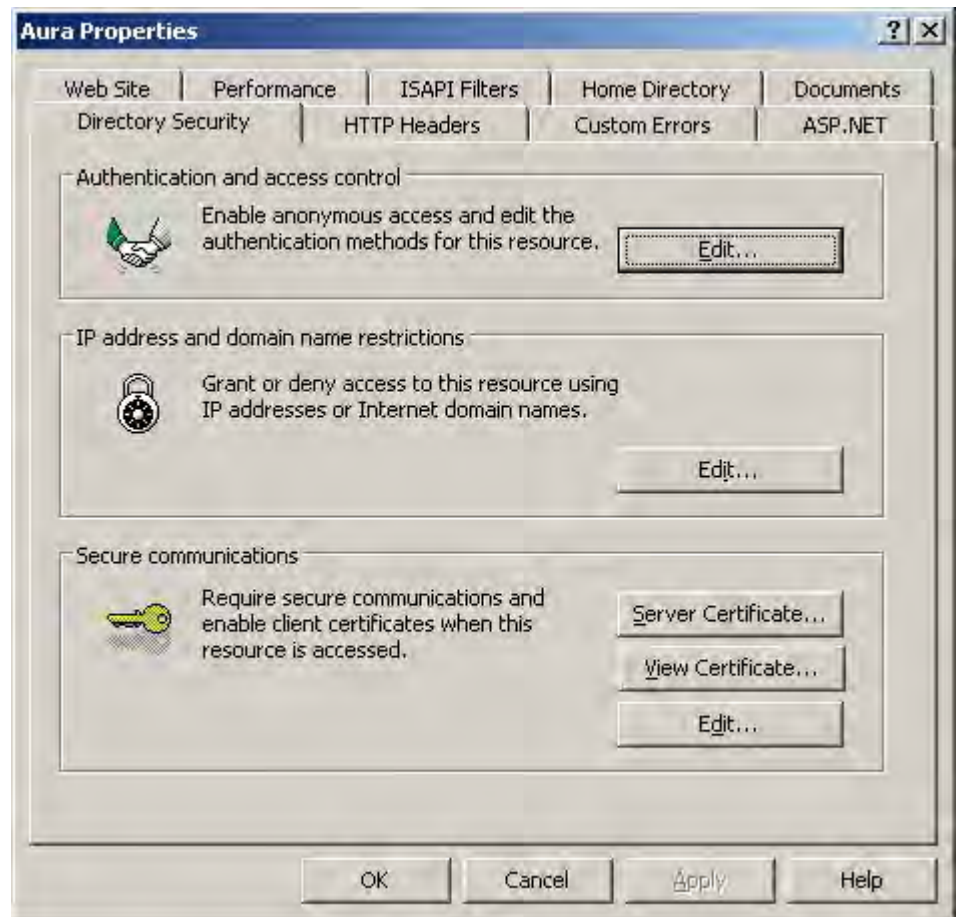
When requesting the certificate from a well-known certification authority like VeriSign, you will obtain only the certificate. When obtained from Tools4ever, you will receive the certificate and the Tools4everCA root certificate.

### **Step 2 - Windows Server 2003: Install the certificate in the IIS website**

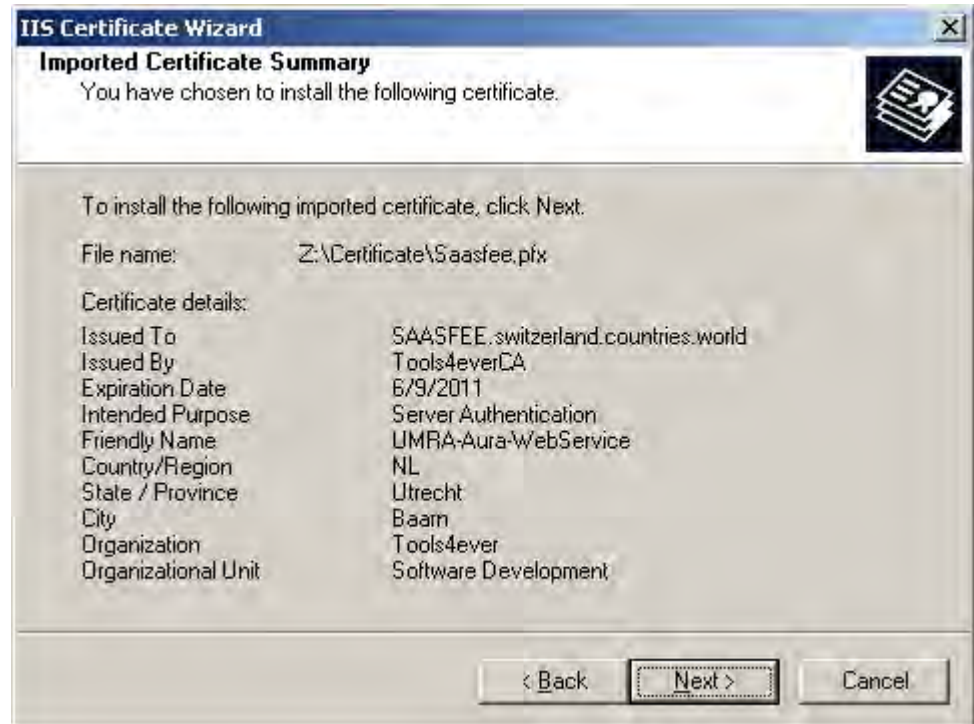
The certificate, including its private key needs to be installed on the website that hosts the webservice. This topic describes the procedure for a computer running Windows Server 2003.

1. Logon to the computer that runs IIS with administrative access;
2. Start **Internet Information Services Manager** and select the website;
3. Select properties and goto tab **Directory Security**;





4. Click on **Server Certificate** and proceed with the **Web Server Certificate Wizard**.
5. When asked for the **Server Certificate**, select option **Import a certificate from a .pfx file**. This is the format used by Tools4ever to provide you with a certificate, including its private key.
6. Specify the name of the certificate file and check option: **Mark cert as exportable**.
7. Check the data shown for the certificate and click **Next**:



8. Finish the wizard. The IIS website can now use the configured certificate.
9. To disable non-secure communications, click option **Edit...** in step 3, check option **Require secure channel (SSL)**, click **OK** and complete the open dialog windows.



10. The IIS web-site is now configured to run in a secure way only.

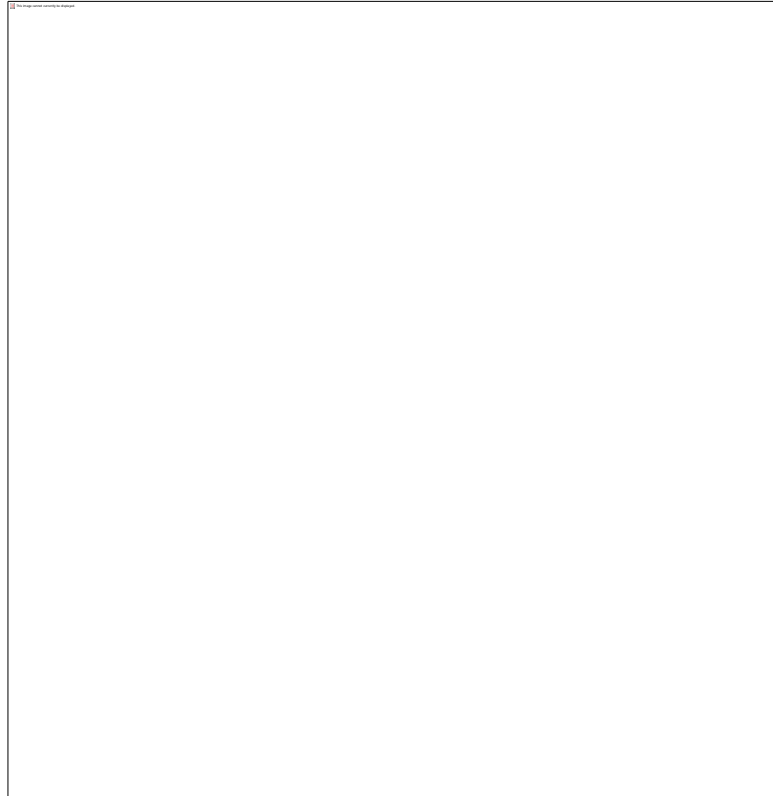
#### Step 2 - Windows XP: Install the certificate in the IIS website

The certificate, including its private key needs to be installed on the website that hosts the webservice. This topic describes the procedure for a computer running Windows XP.

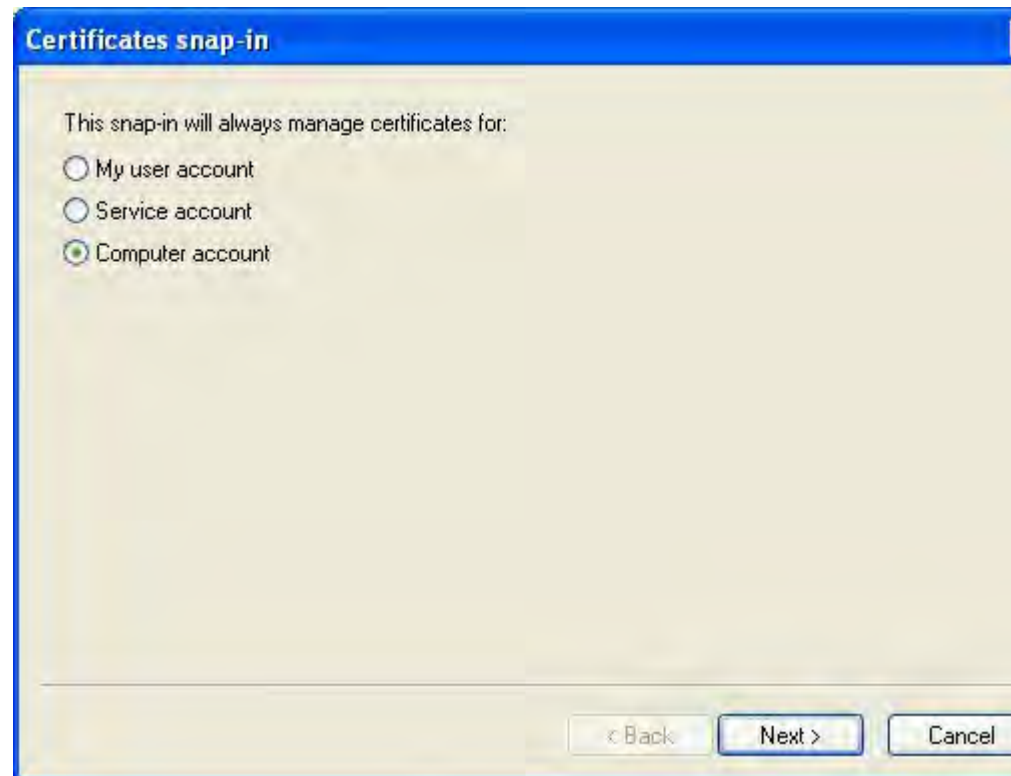
1. Logon to the computer that runs IIS with administrative access;
2. To install the certificate, it must be stored in the local computer store of the machine. This requires the following steps: Select menu option **Start, Run** and enter **MMC**. The Microsoft Management Console is started;
3. Select menu option **Start, Add/Remove Snap-in**. Click **Add** in the dialog shown;



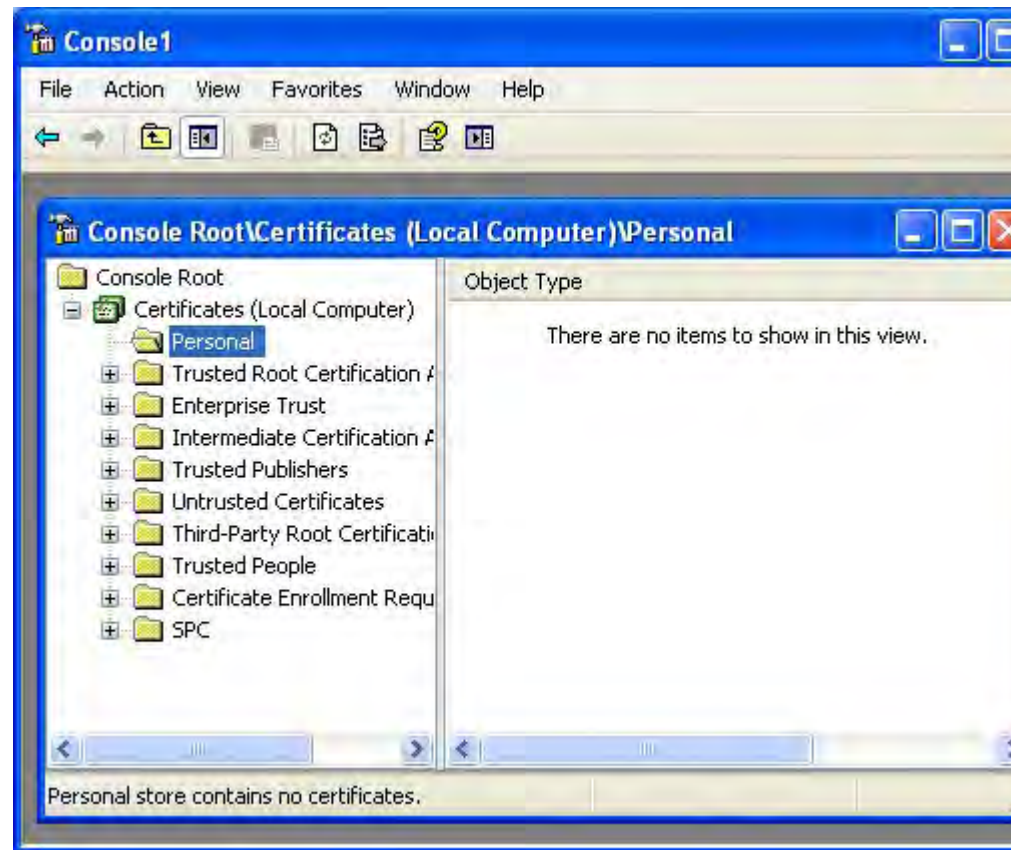
4. Select snap-in **Certificates**. Click **Add**;



5. Select the option to manage certificates for **Computer account**;



6. In the next dialog, select **Local computer** and press **Finish**;
7. In dialog **Add Standalone Snap-In**, click **Close**;
8. In dialog **Add/Remove Snap-In**, click **OK**;
9. To import the certificate, expand the tree on the left and right click on **Personal**. Select menu option **All tasks, Import...**

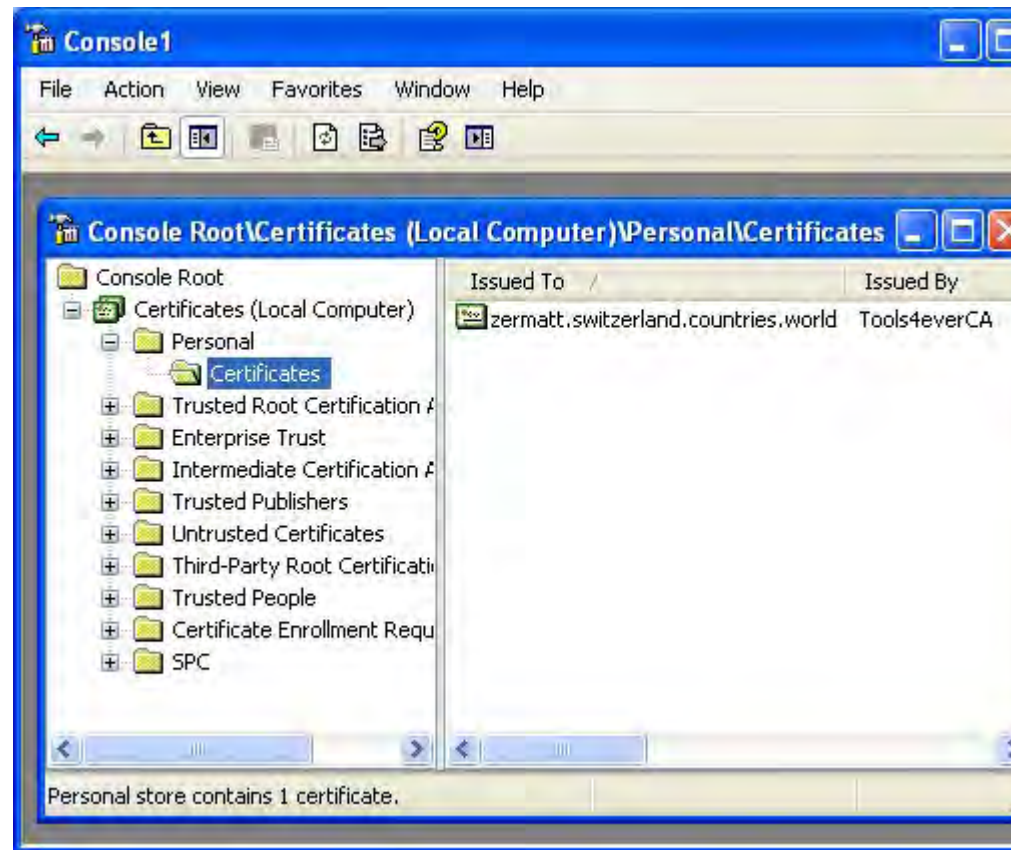


10. The **Certificate Import Wizard** is started. Select the certificate file and proceed with the wizard. Specify the option: **Mark this key as exportable...**

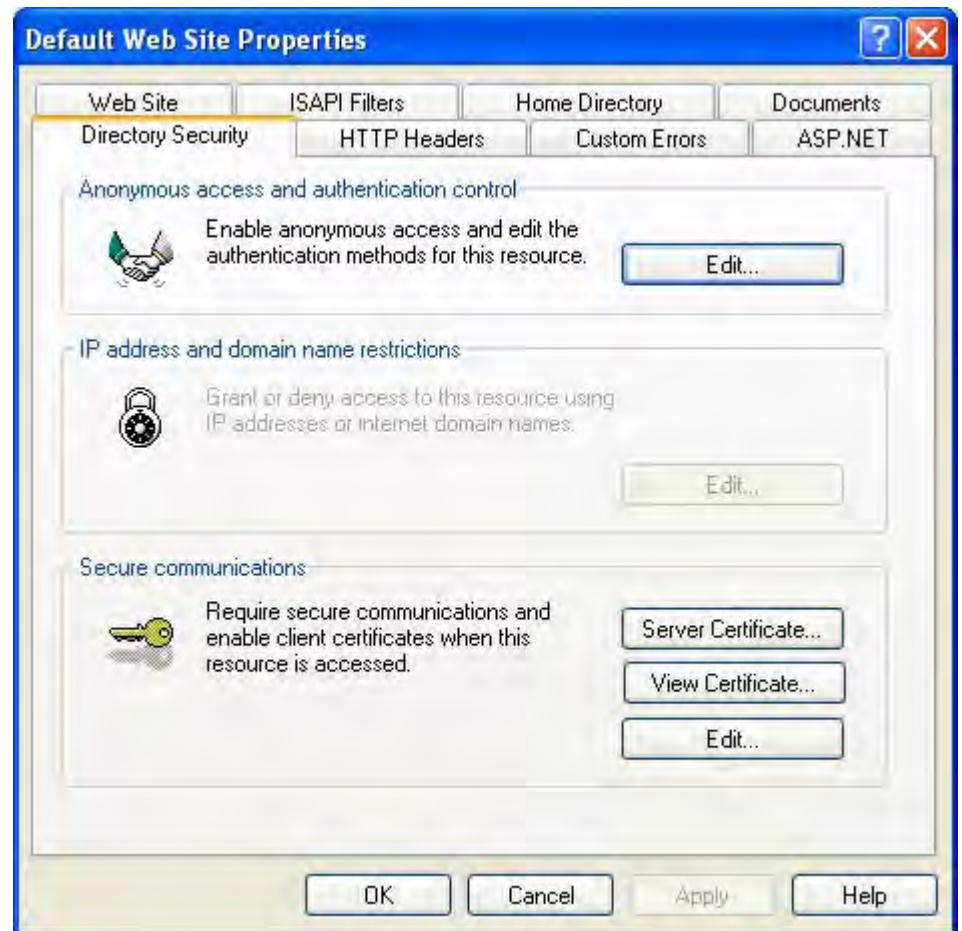


11. When selecting the store, select option **Automatically select the certificate store based on the type of certificate.**
12. Complete the wizard. When done, press F5 to update the tree showing the certificate stores. The certificate should now be shown:





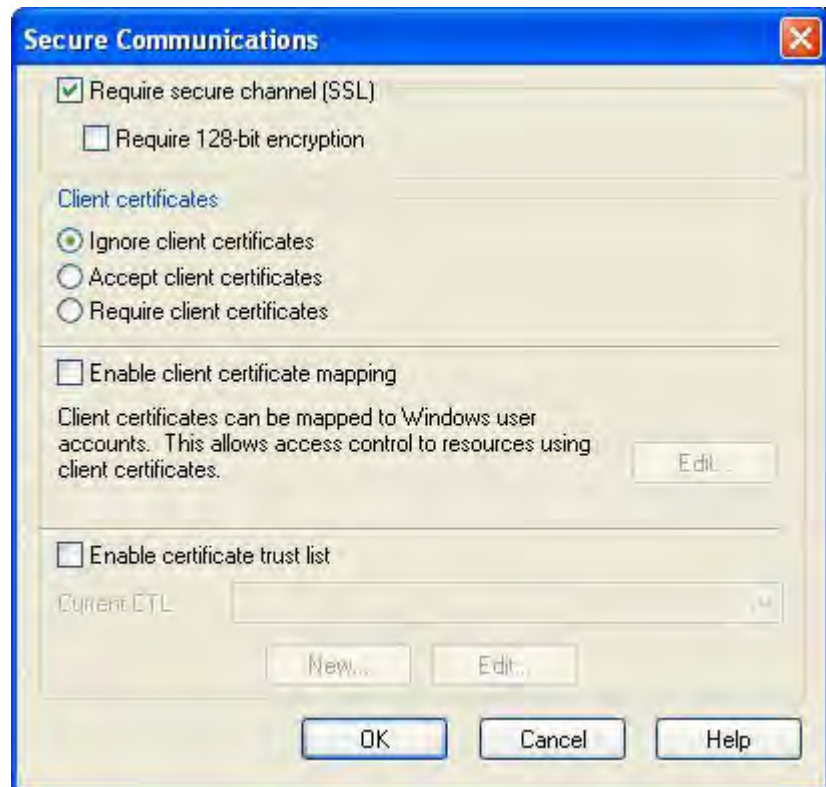
13. Start **Internet Information Services Manager** and select **Default Web Site**. Right click on **Default Web Site** and select menu option **Properties**;
14. Select tab **Directory Security** and click on **Server Certificate**



15. The **Web Server Certificate Wizard** is shown. Proceed with the wizard and select the option to **Assign an existing certificate**. The list with available certificates should show the just installed certificates:



16. Select the certificate and complete the wizard.
17. To disable non-secure communications, click option **Edit...** in step 14, check option **Require secure channel (SSL)**, click **OK** and complete the open dialog windows.



18. The IIS web-site is now configured to run in a secure way only.

### Step 3: Install the Certification Authority root certificate on the UMRA Powershell Agent service computer

**This step is only required if the certificate is obtained from Tools4ever and the Tools4ever root certificate Tools4everCA is not already installed on the computer that runs the UMRA Powershell Agent service.**

When the communication between the IIS Web-Service and the Powershell Agent service is initialized, the Powershell Agent service must trust the certificate offered by the web-service. This is the case if the certification authority that generated the certificate is trusted. Therefore, this root certificate must be installed.

1. Start **Internet Explorer** on the computer that runs the Powershell Agent service;

2. Select menu option **Tools, Internet Options...** and select tab **Content**;
3. Click button **Certificates**;
4. Click button **Import**. The **Certificate Import Wizard** is started;
5. With the wizard, import the Tools4everCA root certificate (Tools4everRoot.cer);
6. When asked, select option **Automatically select the certificate store based on the type of certificate**
7. Complete the wizard.

The computer will now trust all applications that communicate using certificates generated by the certification authority of the just installed certificate

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

### **3.14. UMRA Google Module**

To support Google Apps, the UMRA Google module consists of a number of dedicated UMRA Google actions and a It is possible to fulfill all required Google tasks with the greatest performance Google allows. Also UMRA will minimize the network traffic between UMRA and Google.

#### **3.14.1. Google - Requirements**

To support Google Apps with UMRA, the following requirements apply:

1. To run UMRA projects that use UMRA Google actions, an UMRA license is required.
2. The following Google Apps accounts are supported: Google Apps Premier, Google Apps Educational

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

3. The provisioning API must be enabled in the Google account.

Note: In a previous version, Google support was provided by using the UMRA Powershell Agent service. This is no longer required.

### 3.14.2. Google - Action: Google Setup Connection

#### Google Session ID

Over 30 UMRA Google actions are available to access Google Apps. All of these actions use property **Google Session ID**. This property identifies the connection to a specific Google Apps domain and is generated with UMRA Google action **Google Setup connection**. By default, the **Google Setup connection** action stores the **Google Session ID** in output variable **%GoogleSessionId%**, which is the default value of property **Google Session ID** for all other UMRA Google actions.

#### Accessing multiple Google domains

When multiple google domains are accessed by UMRA, multiple **Google Session ID**'s are used. There is no need to store the **Google Session ID**'s in a (session) variable list. Each time a **Google Session ID** is required, the action **Google Setup connection** should be used instead. This action requires **no performance from Google Apps domain or network resources when called multiple times**. Internally, UMRA will keep track of the actual connections to Google Apps domains. When **Google Setup connection** is called, UMRA will re-use existing connections.

#### Closing Connections

For every time the **Google Setup connection** is executed, the **Google Close connection** action must be executed, even if the **Google Setup connection** generated an error.

#### Cache



When one of the 2 caches is enabled in the **Google Setup connection** the cache is filled and loaded into memory. The cache is shared between all the connections to the same domain, even if those are different scripts. All those scripts will now read and write to and from the cache.

As soon as the **Google Close connection** action is called with the **Google Session ID** of the **Google Setup connection** which enabled a cache. The changes in that cache will be calculated and written to Google. If no other script is using the cache, it will be removed from memory. If another script is still using the cache, the cache will continue to be used.

### **Provisioning Cache**

When enabled the **Google Setup connection** action will load all the user and group accounts into the cache. All user/group actions will now use the cache. The action **Google Change password** will always be written directly to Google, even if there is a cache in place. If the user is still only in the cache and not yet created in Google, the password is only written to the cache.

When the script is only for writing a single change (as with UMRA forms) it can be delaying to enable and load the cache and it should not be enabled for that script.

Also when only contact actions are used in the script, the cache should be disabled.

For password synchronization scripts the cache should be disabled.

### **Contacts Cache**

When enabled the **Google Setup connection** action will load all the contacts into the cache. When the script will not use contacts actions, the cache should not be enabled, else it should, even for UMRA form projects.

### 3.14.3. Google - Connections

Not every Google setup connection results in an actual connection to Google. All the google requests (if they come from the cache or are directed by an Google action) are stored in a queue. The Google engine will execute them in order. If the queue fills up, UMRA will start to execute them in a separate task. While this task is busy, UMRA will execute the next request again in separate task and will look if there are more requests. Up to 10 task can run simultaneously. Every task represents an actual connection to the Google domain. When the number of simultaneous connections to Google is limited in the registry, this also limits the maximum number of task.

### 3.14.4. Google - Registry settings

For analysis and debugging purposes, UMRA supports specific registry settings that affect the operations of Google related functionality. By default, these settings should not be specified. In this case, the default settings automatically apply. In specific circumstances, one can changes these settings to manage the Google operations at a more detailed level. Once changed, the UMRA software, either the UMRA Console or UMRA Service application, **must be restarted** in order for these registry settings to take effect.

#### Registry key

For the UMRA software, the following registry key must be used:

UMRA Service, 32-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Tools4ever\UmraSvc\Config

---

UMRA Service, 64-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Tools4ever\UmraSvc\Config

UMRA Console, 32-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Tools4ever\UmraConsole\Config

UMRA Console, 64-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Tools4ever\UmraConsole\Config

### Registry values

Name	Type	Default value	Description
------	------	---------------	-------------

GoogleFileLoggingMode	REG_DWORD	0	<p>Specification of the logging mode used by the UMRA Google engine.</p> <p>0: Default setting, only errors and main progress information is logged</p> <p>1: Each internal UMRA Google connection will use it's own set of cyclic log files. The log information includes detailed progress information and error information.</p>
GoogleFileLoggingCycleCount	REG_DWORD	2	<p>The number of log files of a single set of cyclic log files.</p>
GoogleFileLoggingCycleSizeMb	REG_DWORD	5	<p>The size in MB of a single file of the set of cyclic log files.</p>

---

---

GoogleMaxTotalConnections	REG_DWORD	10	Specification of the maximum number of simultaneous connections to Google.
---------------------------	-----------	----	--

### 3.15. UMRA SAP module

To support SAP, the UMRA SAP module consists of a number of dedicated UMRA SAP actions and a mechanism to control the connections going to SAP systems. It is possible to fulfill all required SAP tasks and to tune the performance of the connections between UMRA and the SAP host systems.

#### 3.15.1. SAP - Requirements

To support SAP with UMRA, the following requirements apply:

1. To run UMRA projects that use UMRA SAP actions, a separate UMRA license is required. The license enables the UMRA SAP actions.
2. The following SAP systems are supported: SAP NetWeaver 6.20,

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

7.00, 7.10; SAP - R/3 Enterprise (4.7), mySAP 2004 (ECC 5.0);

Note: In a previous version, SAP support was provided by using the UMRA Powershell Agent service. This is no longer required.

### 3.15.2. SAP - Action: SAP Setup connection

#### SAP Session ID

Over 30 UMRA SAP actions are available to access SAP systems. All of these actions use property **SAP Session ID**. This property identifies the connection to a specific SAP host system and is generated with UMRA SAP action **SAP Setup connection**. By default, the **SAP Setup connection** action stores the **SAP Session ID** in output variable **%SapSessionId%**, which is the default value of property **SAP Session ID** for all other UMRA SAP actions.

#### Accessing multiple SAP hosts

When multiple SAP host systems are accessed by UMRA, multiple **SAP Session ID**'s are used. There is no need to store the **SAP Session ID**'s in a (session) variable list. Each time a **SAP Session ID** is required, the action **SAP Setup connection** should be used instead. This action requires **no performance from SAP systems or network resources when called multiple times**. Internally, UMRA will keep track of the actual connections to SAP host systems. When **SAP Setup connection** is called, UMRA will re-use existing connections.

### 3.15.3. SAP - Connections

To prevent errors and performance problems, the number of connections to SAP host systems should be managed carefully.

Depending on the exact configuration, hardware in the system, activity of SAP and other processes, a maximum number of connections to a SAP host system applies. As a general rule, one can say that in a safe scenario no more than 5 UMRA connections should be used to access the same SAP host at the same time and no more than 30 UMRA connections should access all SAP hosts at the same time together.

If only one UMRA project is active, only a single SAP host system is accessed at the same time. But because the UMRA software can run multiple projects at the same time and SAP connections should be re-used to improve performance, multiple projects can access multiple SAP host systems in practice. In order not to exceed the connection limitations, UMRA contains a sophisticated queuing mechanism that will delay all UMRA projects when these limits are to be exceeded. So even if 100 UMRA projects accessing SAP hosts are started at the same time, the connection limitations are not violated. In this case, the execution time of the UMRA projects will increase since projects are delayed.

The connection limits are maintained by the UMRA Service and UMRA Console application and can be changed from their default values. Select menu options **UMRA Service, Service properties, SAP** for the UMRA Service and **Tools, Options, SAP** for the UMRA Console application to specify these limits.

#### **3.15.4. SAP - UMRA SAP child process**

With UMRA, the SAP host systems are accessed using a separate process, the UMRA SAP child process. When running, these processes show up in the task manager. The processes have the name **UMsapCmd.exe** and multiple of these process can exist at the same time. Note that these processes are completely managed by the UMRA software and are required to access SAP systems.

#### **3.15.5. SAP - SAP Generic function module**

A special UMRA action is available to support SAP RFC / BAPI function modules that are not available through the other UMRA SAP actions. With this action, any SAP function module can be implemented with UMRA. To support the SAP function module, the function module name and all import and export parameters must be specified. All this information is available from the BAPI Explorer that can be accessed for instance with the SAP NetWeaver client.

#### **UMRA action**

In UMRA, the action is available from the action tree: **SAP, General, SAP Generic function module**. For this action, the following fields must be specified:



**SAP Session ID variable**

A variable that identifies the session that connects UMRA with SAP. The variable is the result of action **SAP Setup connection**.

**Function module**

The name of the SAP function module. Example

BAPI\_USER\_EXISTENCE\_CHECK

**Input parameters**

Specify all input parameters for the function module. For each input parameter, specify the parameter name as used in SAP and the type and value. All parameter values can be specified using UMRA variables but the **Text** and **Numeric** parameter values can also be specified directly. The input types **Date**, **Structure** and **Table** must be specified using an UMRA variable. The **Date** value must be specified using an UMRA date-time variable. The **Structure** type value must be specified using an UMRA table variable holding a table with a single row. The table column names must correspond with the structure fields. Only the fields with input values need to be specified as table columns. Other fields can be omitted. The **Table** type value must be specified with an UMRA table. In SAP, a table is an array of equal typed structures. The UMRA table columns must correspond with the SAP structure fields. Only the fields with input values need to be specified.

**Output parameters**

Specify all required output parameters using output variable names. For each output parameter, specify the correct type: **Text**, **Numeric**, **Date**, **Structure** or **Table**. For **Structure** and **Table** output parameters, UMRA will generate an UMRA table. The UMRA table for the SAP structure type will hold only a single row. The structure field names correspond with the UMRA table column. In case not all structure fields or table columns need to be exported from SAP, an existing empty UMRA table variable can be specified. The UMRA table should contain one or more specified columns but no rows. In this case, only the fields or columns for which an corresponding UMRA table column exists are exported from SAP.

**Error handling - Check RETURN structure or table for errors. When found, raise an UMRA action error event.**

Many SAP function modules use the RETURN structure or table to export results. The RETURN structure or table can contain error information in case something goes wrong. In such scenarios, the execution of the SAP function module is successful, but nevertheless, an error has occurred. If

you want UMRA to recognize such an error as if the UMRA action failed, check this option. When the option is checked, UMRA will analyze the RESULT parameter and raise an error event when the RETURN parameter contains an error. **In this case, it is required that the RETURN parameter is specified as one of the output parameters.**

#### 3.15.6. SAP - Example projects

UMRA contains a number of example projects that show how to use the UMRA SAP actions that can be used in UMRA projects. The example projects can be found in the following location:

[UMRA Console program dir]\Example Projects\SAP

By default, the location path is:

C:\Program Files\Tools4ever\User Management Resource Administrator\Example Projects\SAP

#### 3.15.7. SAP - Registry settings

For analysis and debugging purposes, UMRA supports specific registry settings that affect the operations of SAP related functionality. By default, these settings should not be specified. In this case, the default settings automatically apply. In specific circumstances, one can change these settings to manage the SAP operations at a more detailed level. Once changed, the UMRA software, either the UMRA Console or UMRA Service application, **must be restarted** in order for these registry settings to take effect.

##### Registry key

For the UMRA software, the following registry key must be used:

---

UMRA Service, 32-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Tools4ever\UmraSvc\Config

UMRA Service, 64-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Tools4ever\UmraSvc\Config

UMRA Console, 32-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Tools4ever\UmraConsole\Config

UMRA Console, 64-bit OS:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Tools4ever\UmraConsole\Config

### Registry values

Name	Type	Default value	Description
------	------	---------------	-------------

SapFileLoggingMode	REG_DWORD	0	<p>Specification of the logging mode used by the UMRA SAP child process.</p> <p>0: Default setting, only errors and main progress information is logged</p> <p>1: Each UMRA SAP child process will use it's own set of cyclic log files. The contents of this file is also returned to UMRA and shown in the UMRA log file. The log information includes detailed progress information and error information.</p>
SapFileLoggingCycleCount	REG_DWORD	2	<p>The number of log files of a single set of cyclic log files.</p>
SapFileLoggingCycleSize	REG_DWORD	1	<p>The size in MB of a single file of the set of cyclic log files.</p>

SapProcessMode	REG_DWORD	0	<p>Specification of the SAP process mode.</p> <p>0: Normal setting</p> <p>1: No SAP access: No SAP systems are accessed. This setting should be used for debugging purposes only.</p>
SapJobDelayMin	REG_DWORD	0	<p>Specification of the minimum SAP process delay in seconds. Each time when UMRA activates the UMRA SAP child process, the child process will wait for at least the specified number of seconds.</p>
SapJobDelayVar	REG_DWORD	0	<p>Specification of the maximum variation of the SAP process delay in seconds. Each time when UMRA activates the UMRA SAP child process, a random value between 0 and the specified number is generated. The effective delay equals the value of SapJobDelayMin and the generated value.</p>

### 3.16. AFAS Online

By default AFAS will not return data to UMRA. To enable this, certain **GetConnectors** must be imported into the AFAS GetConnector console. These can be found in the **User Management Resource Administrator\Dynamic Actions** folder and have a .gcn file extension. It is recommended to have the AFAS system administrator import these files into their AFAS environment.

#### 3.17. Setup connection

To make a connection to AFAS Online a user must be available in the AFAS Authorization tool and the connector option must be enabled. The following variables must be entered

- AFAS Username – The username of the user created in the authorization tool
- AFAS Password – the password of the user created in the authorization tool
- AFAS Environment – The name of the customers AFAS environment. Which by default has the following layout “OXXXXXAA” where X is a numeric value.

### 3.18. AFAS get employees

is action returns all employees divided in two different tables. One for active and one for inactive employees. The difference of active and inactive employees is determined by their employment end date (if entered).

By default this action only returns the active employees of that moment. This can be changed by entering a date/time variable in the active reference date. This will return all current active employees and all employees that will enter service between the current date and the reference date.

### **3.19. AFAS get employees contract**

This action returns a table with active contracts of all employees. There following variables can be entered/modified to change the outcome.

- Active reference date – Return all active contracts of the given date/time variable.
- Active only – By default this is yes, by changing this variable to no all contracts are returned. Including past or future contracts.

### **3.20. AFAS Get organigram**

This action returns a table with the organization organogram. Not all profit users will have the organogram entered in AFAS Online. So this is only relevant for customers who use this data to determine managers for example.

### **3.21. AFAS Export Date**

This action allows you to receive data by a custom created “getConnector”. The following variables can be entered/modified to change the outcome.

- Connector name – The name of the AFAS getConnector. For example “umra\_get\_employees”.
- Filter XML – The Filter XML can be determined by starting the GetConnector item in the Profit interface and enter the filter details. Click the Filter XML button to view the generated filter XML. Copy the xml code into this field.
- Include meta data – This option will return the meta data as specified in profit.
- Export as XML – Choose to export data as comma separated file or as XML. Default is “No”;
- Output options – Choose how data is formatted and separated;
- File name – A variable where the data must be stored. The filename is optional. If not specified the action will determine an unique filename in the temporary directory of the Powershell Agent Service account and create the file. The output variable will be filled with this filename. Alternatively a location path such as “c:\program files\umraservice\afasdata.csv” can be entered;

### **3.22. AFAS Update employee**

This action allows UMRA to write data back to AFAS Online. One of the following input fields are mandatory.

- Employee ID – The employee of the user that must be modified;
- BSN – The social security id of the user that must be modified;
- Employee Code – The employeecode of the user that must be modified (can be different from the employeeID).
- UMRA can update the following fields
  - Work phone
  - Private phone
  - Work mobile
  - Private mobile
  - Work e-mail
  - Private e-mail

If fields are left empty they will not be overwritten.



---

### 3.23. Password Synchronization Manager

### **3.23.1. Goal**

The UMRA Password Synchronization Manager ( PSM) is a piece of software that detects password changes of Windows Active Directory domain accounts. In conjunction with an UMRA service, it allows for the propagation of new or modified passwords to other domains or systems.

### **3.23.2. Installing UMRA PSM for the first time**

This section will provide a step by step description of installing and configuring the PSM software for first use in your network.

#### **Prerequisites**

1. In order to start with UMRA PSM, there must exist an operational UMRA console on your local computer, with a live connection to the UMRA service in your network.

*Copyright © 1998 - 2011, Tools4ever B.V.*

*All rights reserved.*

*No part of the contents of this user guide may be reproduced or transmitted in any form or by any means without the written permission of Tools4ever.*

*DISCLAIMER - Tools4ever will not be held responsible for the outcome or consequences resulting from your actions or usage of the informational material contained in this user guide. Responsibility for the use of any and all information contained in this user guide is strictly and solely the responsibility of that of the user.*

*All trademarks used are properties of their respective owners.*

2. You must be logged in at the UMRA console with an account with full administrative rights to the domains for which you want to use UMRA PSM.
3. The UMRA Service must have the correct license installed that allows the use of the PSM module.

### **Overview**

The UMRA PSM software consists of three distinct parts:

1. A Notification package that is installed on each domain controller on the domain. This package is notified automatically by Windows on each password modification against the domain controller. The package contacts a UMRA service to report all changes.
2. An extension to the UMRA service, which listens to the Notification events, and will execute a specified UMRA project in response, that may synchronize the passwords to other systems, or perform various other network related actions.
3. A general management module in the UMRA Console that is used to install, configure and monitor the PSM software.

Installation is performed in two main steps.

1. Installation of the Notification package on each domain controller in the domain.
2. Configuration of the UMRA service to act on Password change notifications.

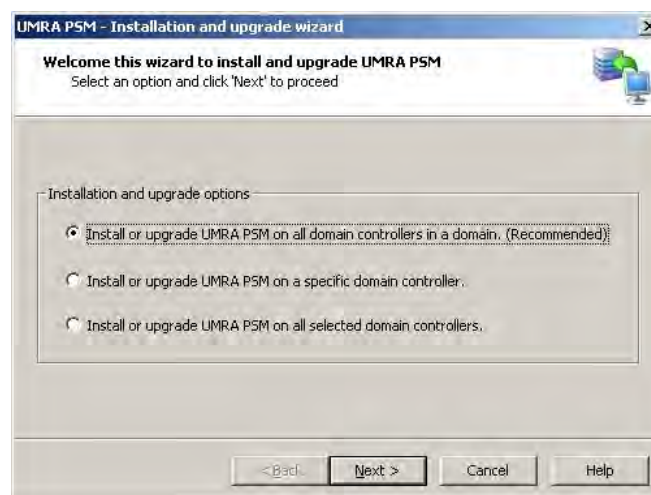
### **Installation of the Notification Package**

In order to install the Notification package on each domain controller, perform the following steps:

1. Log in at your pc with an account with domain administrative rights.
2. Start the UMRA console.

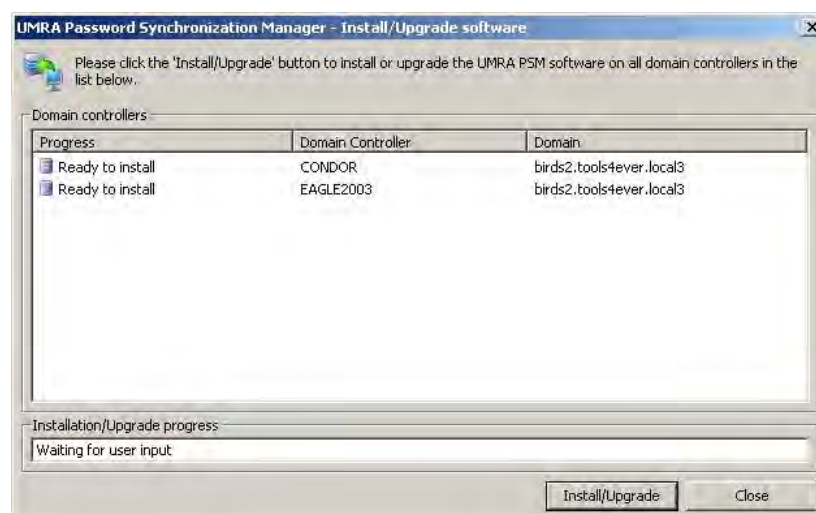
3. Verify that the console is connected to the UMRA service.
4. Choose the menu command **Tools, Password Synchronization Manager, Installation - Upgrade**.

The following window will show:



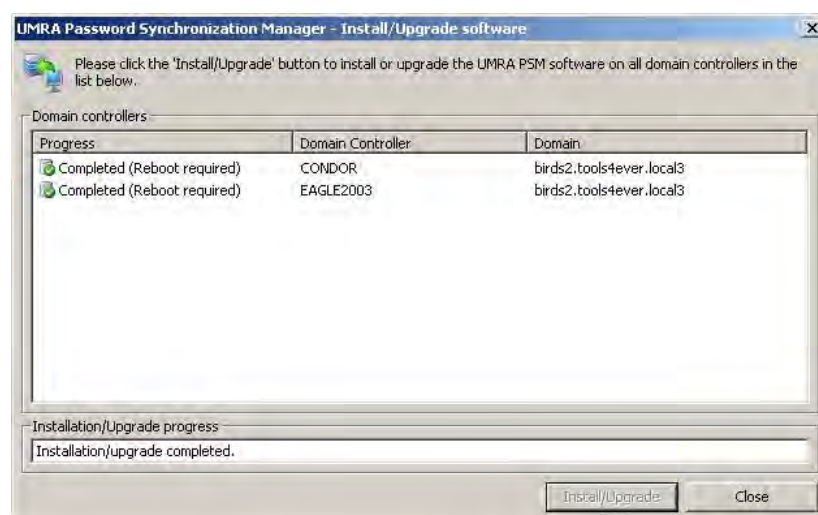
In order to guarantee that all password changes in a domain are caught, the PSM package must be installed on each domain controller. Choose the first option, and select **Next**.

5. In the following screen, you are prompted for the domain name. Enter the full DNS domain name of the domain, and press **Next**.
6. The network will be searched for all domain controllers, and the following screen will be shown.



Next select **Install/Upgrade** to actually install the PSM package on the domain controllers.

7. The results of the installation are shown; optionally you can look in the UMRA console log for details.



Select close to go to the main PSM overview window.

- 8.

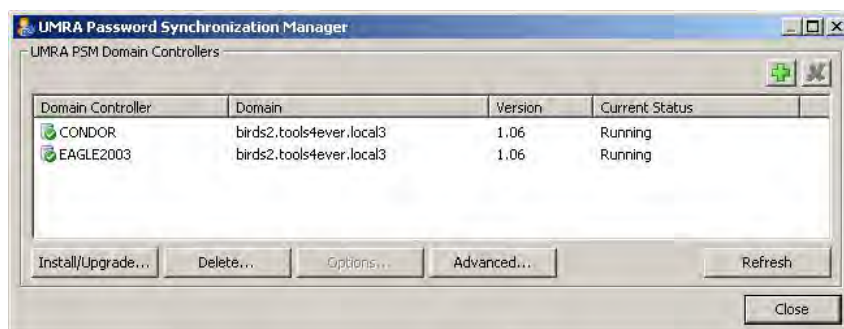


Select close to exit the setup.

9. Reboot all domain controllers in the domain.
10. After the reboot, go to **Tools, Password Synchronization Manager, Installation - Upgrade** to view the current status.

If not all servers have entirely completed reboot, some servers may show as unavailable or report access denied. Wait until they are operational again, and select the **refresh** button to update the status in the overview.

11. When all domain controllers are listed as running, the PSM package is operational.



Note, that the status "running" means that the package is installed correctly, and loaded successfully by Windows, and is configured to contact the UMRA service when a password set or change event occurs. It does not imply success or failure of any particular notification. By default, the package will log its operations in the file UmraPsm.log, located in the system32 directory of the domain controller.

12. The installation of the package is now complete, select close to exit.  
Next step is to configure the UMRA service to act on received notifications.

#### Configuration of the Umra Service

The UMRA service will execute a specific project for each password change notification it receives from the PSM package.

1. Create a new empty project that must be executed. The required project is a standard automation project, it can be created by means of **File, New, Automation Project**.
2. **Select Tools, Password Synchronization Manager, Configuration** and specify the just created project as the project that must be executed.



The specified project will be executed on a notification from the PSM package.

If the enable flag is switched off, the service only will acknowledge in the log (when the appropriate loglevel is enabled), that a PSM notification has been received by the service, but no project will be executed.

3. Within this project you must specify the script actions to execute as in any other project. The domain name, account name and new password value received from the PSM package are available in script variables at the start of the script. They can be used in the script actions, for instance to update the password in other systems.

A full overview of those script variables that have a value at the start of the script is listed in *UMRA PSM Script variables* on page 127. The project log file is located in the log directory of the UMRA service, and called UmraPsm1.txt.

**Warning:** Take care with actions that create accounts or update passwords in one of the domains for which PSM is active, as such actions may themselves cause a new password change event notification. Always use a one way synchronization (system A to system B), and make sure that there is no synchronization active in the other direction (system B to system A).

### 3.23.3. Miscellaneous UMRA PSM topics

#### Managing UMRA PSM

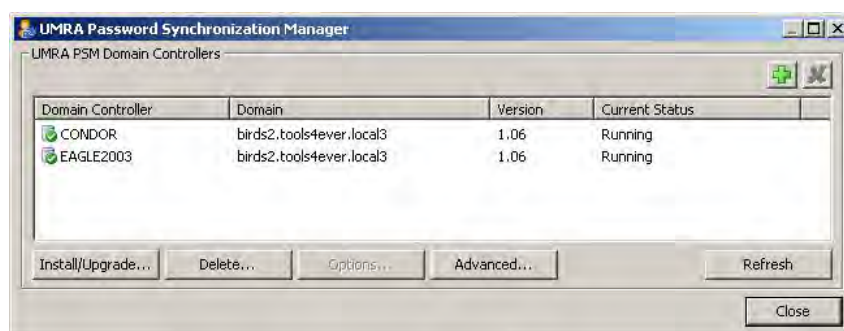
1. To view and manage the installation of the UMRA PSM Notification packages , select **Tools, Password Synchronization Manager,**

**Installation - Upgrade.** Next press **F1** to get help on the available options.

2. To specify the UMRA automation project that should be executed on a received notification event , select **Tools, Password Synchronization Manager, Configuration.**
3. To specify the Actions that UMRA should take on a password reset notification, modify the specified automation project. The project is called once for each individual password change. see *UMRA PSM Script variables* on page 127 for the available special variables that have a value when the project is called.

### Viewing the current status of PSM Packages

To get an overview of the current status of the PSM Packages installed on the domain controllers in your domain select the menu command **Tools, Password Synchronization Manager, Installation - Upgrade.** This will display an overview like the image below.



The PSM packages must have the single status **Running** on all domain controllers that are online, in order to intercept all change password events for the specific domain.

See *UMRA PSM installation status* on page 126 for the possible states listed.

Note, that the status "running" means that the package is installed correctly, and loaded successfully by Windows, and is configured to contact the UMRA service when a password set or change event occurs. It does not imply success or failure of any particular notification. By



default, the package will log its operations in the file UmraPsm.log, located in the system32 directory of the domain controller.

#### *UMRA PSM Package installation status*

The PSM Notification packages must have the single status **Running** on all domain controllers that are online, in order to intercept all change password events for the specific domain.

A combination of some of the following states is possible:

#### **Running**

The package is installed correctly, and loaded successfully by Windows, and it is configured to contact the UMRA service when a password set or change event occurs. It does not imply success or failure of any particular notification.

#### **Upgrade required**

The installed version of the Package is lower than the version that ships with your current version of UMRA. To ensure proper operation the package on the domain controller should be upgraded.

#### **Reboot required**

A reboot of the domain controller is required to complete Installation or deletion.

#### **Notification switched off**

The package on the specific domain controller is explicitly configured not to send any notification to the UMRA service.

#### **Installed**

The PSM package is installed, but not (yet) running.

#### **Not installed**

The PSM package is not installed on the domain controller.

#### **Computer not available**

The specific domain controller does not respond, it may be offline.

#### **Access Denied**

The UMRA console does not have sufficient rights to connect to the domain controller, or the domain controller is not completely online yet after a reboot.

#### Unknown

The state is not yet determined.

#### Error

General error retrieving the status.

#### UMRA PSM script variables

When the UMRA service is notified by the PSM package of a password change, the designated UMRA automation project will start.

Several special UMRA script variables are set by the service. Their values may be utilized by the script of the Umra project to determine the actions to perform.

Here is a list of all special script variables present at the start of the UMRA project started through a PSM change password notification.

#### Password related variables

Variable Name (with example value)	Explanation
%PsmDomain%=BIRDS2	The (NETBIOS) name of the domain of both the domain controller and the account to change.
%PsmComputer%=EAGLE2003	The (NETBIOS) computer name of the domain controller at which the password was modified.
%PsmAccount%=polymem	The SAM Account Name of the account of which the password was changed.
%PsmPassword%=!testjng1234	The new value of the password.
%PsmAccountRid%=3319	The RID (relative ID) of the account.

### Umra PSM package variables

The variables below contain statistical information on the PSM package on the particular domain controller.

Variable Name (with example value)	Explanation
%PsmDllVersion%=106	The version of the PSM package(dll) that has sent the notification to the UMRA service.
%PsmStatTimeStartup%=07:34 05/28/2008	The system time at which the PSM package was initialized by the Windows LSA process.
%PsmStatTimeRequest%=10:23 05/28/2008	The system time at which the PSM package received the most recent changed password from Windows. Note that this is not necessarily the time at which the specific password of the current user was received by the PSM package.
%PsmStatPasswordChangesCount%=1	The total number of times the PSM package has received a changed password for any user from Windows since startup, at the time %psmStatTimeRequest%.
%PsmStatPasswordSyncRequestCount%=1	The total number of times the PSM package has initiated notifying the UMRA service of a new password since startup.
%PsmStatThreadCount%=1	The current number of threads in the PSM package. This is an indication of how much notifications are queued in the PSM package to be send to the UMRA service.
%PsmStatMaxThreadCount%=1	The maximum number of threads that occurred in the PSM package.
%PsmStatErrorCount%=0	The total number of errors that have occurred in the PSM package since startup.

### UMRA PSM Package registry values

There are several values in the Windows registry on the domain controller that affect the operation of the UMRA PSM Package.

Generally there is no need to change any of them directly by means of the registry editor, and direct modification is strongly discouraged. They are either set automatically during installation, or by modifying configuration settings through the **UMRA console**. ( select **Tools, Password Synchronization Manager, Installation - Upgrade** to open the overview window, select the DC's from the list, and select **Options**)

However, a list of registry settings may be helpful during troubleshooting, and allows for the change of a few rarely modified settings that cannot be changed through the UMRA console.

### Values affecting Registration with Windows

**Key:**

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa

**ValueName:** Notification Packages

**Value Type:** REG\_MULTI\_SZ

**Example Value:** RASSFM KDCSVC WDIGEST scecli UmraPsmW32

**Explanation:** The name of the package is added to the existing values automatically when installing the package with the UMRA console. This instructs the LSA service to try to load the associated .dll on startup.

### Values affecting package behaviour.

All following values are located under

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Tools4ever\UmraPsm** on 32 bit OS domain controllers, and under

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Tools4ever\UmraPsm** on 64 bit OS domain controllers.

ValueName	ValueType	Default Value	Explanation
EnableLogging	REG_DWORD	1	<p>Possible values 1 (default) or 0.</p> <p>1 indicates that logging to a file is enabled. 0 indicates that nothing is logged at all. Set by configuration options in the UMRA console.</p>
LogFile	REG_SZ	<ValueName not defined>	Optional, the full local path to the log file. If not specified, the system32\UmraPms.log is used.
LogFileSize	REG_SZ	<ValueName not defined>	Optional, the maximum size in bytes of the log file. When this value is exceeded, the file is cleared, and logging will continue. If not specified the maximum size of the file is 5 MB.
LogLevel	REG_DWORD	7	<p>Value determines what kind of messages are logged. 7 indicates a combination of: Error (1), warning (2) and informational (4) messages.</p> <p>Set by configuration options in the UMRA console.</p>
NotifyEnabled	REG_DWORD	1	<p>Possible values 1 or 0.</p> <p>If set to 1 (default), the UMRA service will be contacted when a password change is detected.</p> <p>If set to 0, no attempt to contact a UMRA service is made.</p> <p>Set through configuration options in the UMRA console.</p>

Operational	REG_DWORD	0 or 1	System only. Do not change manually. Value is set to 1 by the Package itself when the Windows Isass process initializes the package. Is used to detect that the package is loaded.
PortNumber	REG_DWORD	56814	TCP/IP port number at which the UMRA Service can be contacted. Automatically set during installation by the UMRA console, with the value of the service the console was connected to at that time.
ServerList	REG_SZ	<ValueName not defined>	Optional. If specified it overrules the ServerName setting.  Contains the list of UMRA services that should receive notifications. The first server is contacted by default. if unresponsive, the next server in the list is contacted instead.  format:  "servername:portnumber,servername:portnumber"  Set by configuration options in the UMRA console.
ServerName	REG_SZ	NETBIOS name of the server that runs the UMRA service	The name of the server at which the UMRA service runs that should receive notifications. Automatically set during installation by the UMRA console, with the value of the service the console was connected to at that time.

#### 3.23.4. Manage Active Directory with the UMRA Powershell Agent service

A number of UMRA dynamic actions require the **Exchange 2007 Management Tools** to be installed, although these actions do not manage Exchange 2007 related resources. For these actions, it is not necessary to have Exchange 2007 Server installed on any server, but the Exchange 2007 Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service. Examples of these UMRA actions are: **Get AD permissions** (action folder: Powershell, Active Directory permissions) and **Get (nested) group memberships** (action folder: Powershell, Group management). These actions are implemented using cmdlets that are part of the Exchange 2007 *snap-in* **Microsoft.Exchange.Management.PowerShell.Admin**. Therefore, the Exchange Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service in order to use these actions. Note that not all cmdlets of this snap-in can be used if Exchange 2007 server is not installed. For instance, the cmdlets to create a mail-enable user account requires Exchange 2007 Server to be installed.

The described UMRA dynamic actions have the following characteristics:

1. The actions use cmdlets that are part of the Exchange 2007 Powershell snap-in that comes with the Exchange 2007 Management Tools;
2. The cmdlets do not require Exchange 2007 Server to be installed in the network. Instead, the cmdlets can be used to manage Active Directory resources;
3. To execute the Powershell scripts that use these cmdlets, the Exchange 2007 Management Tools need to be installed on the computer that runs the UMRA Powershell Agent service.

The Exchange 2007 Management Tools are available for 32-bit and 64-bit platforms. When Exchange 2007 Server is installed (64-bit platform only), the tools are installed automatically. To install the tools on a 32-bit platform, see *Setting up the Exchange 2007 Management Tools on a 32-bit platform* for more information. To install the tools on a 64-bit platform without installing Exchange 2007 Server, a similar procedure must be used.

## 3.24. Education

UMRA supports a number of **connectors** for educational systems. For each of these systems, a collection of UMRA actions is available. Each action implements a specific task for the system. The connectors consists of the specific UMRA actions for each system.

### 3.24.1. Aura connector installation

Aura is a Dutch company that offers library software that is primarily used in schools. The UMRA connector for Aura integrates the student information system, for instance Magister, @VO or nOISe, with the Aura software. Changes in the student information system are automatically propagated to Aura.

The connector uses SOAP to integrate the different systems. The central component is the UMRA-Aura-Webservice that communicates with Aura and is accessed by the UMRA software to implement the connector.

The installation of the UMRA-Aura connector includes a number of steps. These are described in the following topics.

#### **Aura installation 1 - Architecture**

Both the UMRA and Aura software are available in different versions and modules. It is important to understand how the connector operates in order to install the software successfully. The following software modules implement the connector:

1. **Aura standard software**  
The Aura standard software can consist of a number of modules. In all cases, the following modules are installed **Aura Catalogus** and **Aura Uitleen**. These are the conventional applications that are already in use without the UMRA-Aura connector in place.
2. **UMRA-Aura-WebService**  
This is the central component that interfaces between UMRA and Aura. The web service runs on top of Microsoft's Internet



Information Service (IIS) and communicates according to the SOAP standard. The web service receives requests from the UMRA software using SOAP and handles these requests by accessing the Aura standard software environment (Aura proprietary). A response from the Aura software is send back to UMRA. In the UMRA project script, the response can be used for further processing. Each operation of the connector always starts with UMRA executing a script action that results in a SOAP request that is handled by the web service. The installation procedure mainly deals with setting up this web service.

3. **UMRA Powershell Agent service**

The UMRA Powershell Agent service is the UMRA module that accesses the UMRA-Aura-WebService. The service converts the UMRA script actions to Powershell code that communicates with the web service using SOAP.

4. **UMRA Service or UMRA Mass**

The UMRA modules that executes project scripts contains actions that manage Aura.

Note that these modules can all run on different computers but also on one and the same computer.

**Aura installation 2 - Prerequisites**

The following prerequisites apply for the different software modules.

Module	Prerequisites	Remarks
Aura standard software	Version 9 or higher	

UMRA-Aura-WebService	Operating system:  Windows XP  Windows Server 2003	The UMRA-Aura-WebService runs on top of IIS (Internet Information Services). IIS must be installed on the computer. It is <b>required to have ASP.NET 1.1.432</b> installed on IIS. With other versions of ASP.NET, for instance <b>ASP.NET 2.x or higher, the connector will not function</b> . When running the UMRA-Aura-WebService on a server platform (Windows Server 2003) an Aura client/server license is required.
UMRA Powershell Agent service	Build 1573 or higher	
UMRA Mass / UMRA Service	Build 1573 or higher	

### Aura installation 3 - Create user account

There are a number of ways to setup how the UMRA-Aura-WebService on IIS accesses the Aura database server. In this documentation, a dedicated, new domain account is used. This is the most transparent approach for all operating systems but other methods may work as well. For this method, it is required that all computers that run (1) the standard Aura software, (2) UMRA-Aura-WebService, (3) UMRA Powershell Agent service and (4) UMRA are member of the same Windows domain. If this is not the case, the installation may differ from the procedure described in this documentation.

In the domain, create a new account and remember the password. Configure the password settings so that it will never expire.

Example:

Username: UmraAuraUser

Password: secret3635

#### Aura installation 4 - Create web site

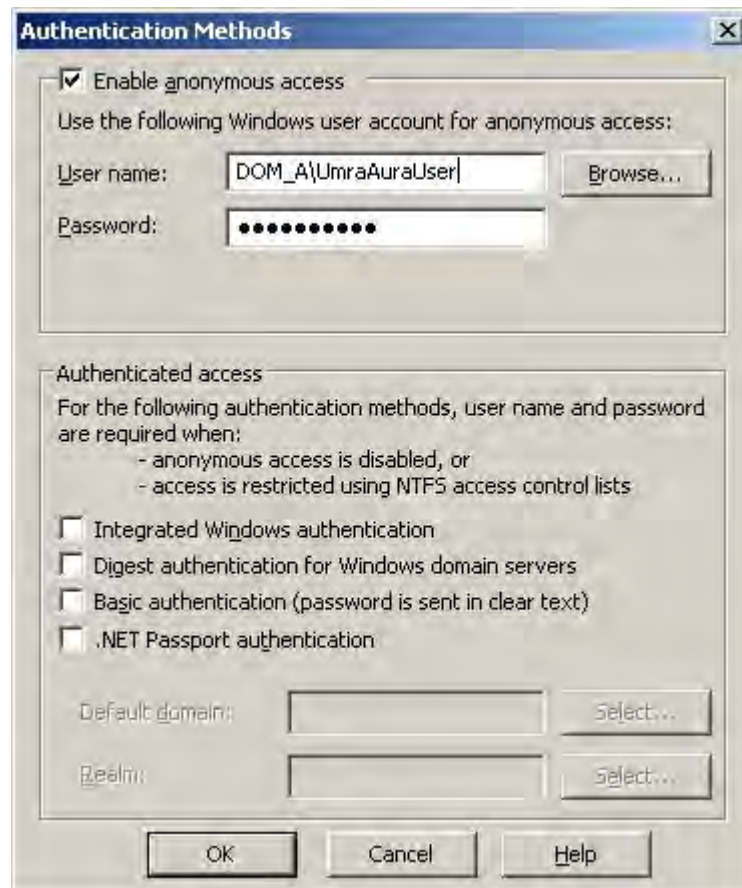
This topic describes how to setup the web-site that hosts the UMRA-Aura-WebService.. To simplify the procedure, the web-site is setup not-secure. To setup the web-site secure, see *Configuring a secure web-site with IIS* for more information:

1. To start, log on to the computer that is going to run the **UMRA-Aura-WebService** with administrative access. On this machine, IIS should be installed. Note that Windows XP and Windows Server 2003 use different versions of IIS, 5.1 and 6.0 respectively;
2. Download the UMRA-Aura-WebService files from **<http://www.aura.nl/aura90/UmraAuraWebservice9.zip>**;
3. Start **Internet Information Services ((IIS) Manager** and create a new web-site for the UMRA-Aura-WebService. In order to access the web-site using the host name only, other web-site hosted on the same computer should not use port 80.
4. The web-site needs at least **read** access to the **local path** and **execute permissions** for **scripts** as shown in the following picture.



During the creation of the web-site a directory is created. By default, the location of a web-site is a subdirectory of the IIS directory, for instance C:\InetPub\Aura.

5. Unpack the UMRA-Aura-WebService files to the newly created directory;
6. Specify the account of the web-site used when accessing Aura:  
Select the properties of the newly created web-site, select tab **Directory Security**, section **Authentication and access control**. Click **Edit**. Uncheck all options, except **Enable anonymous access** and specify the username and password of the account created in the previous step:



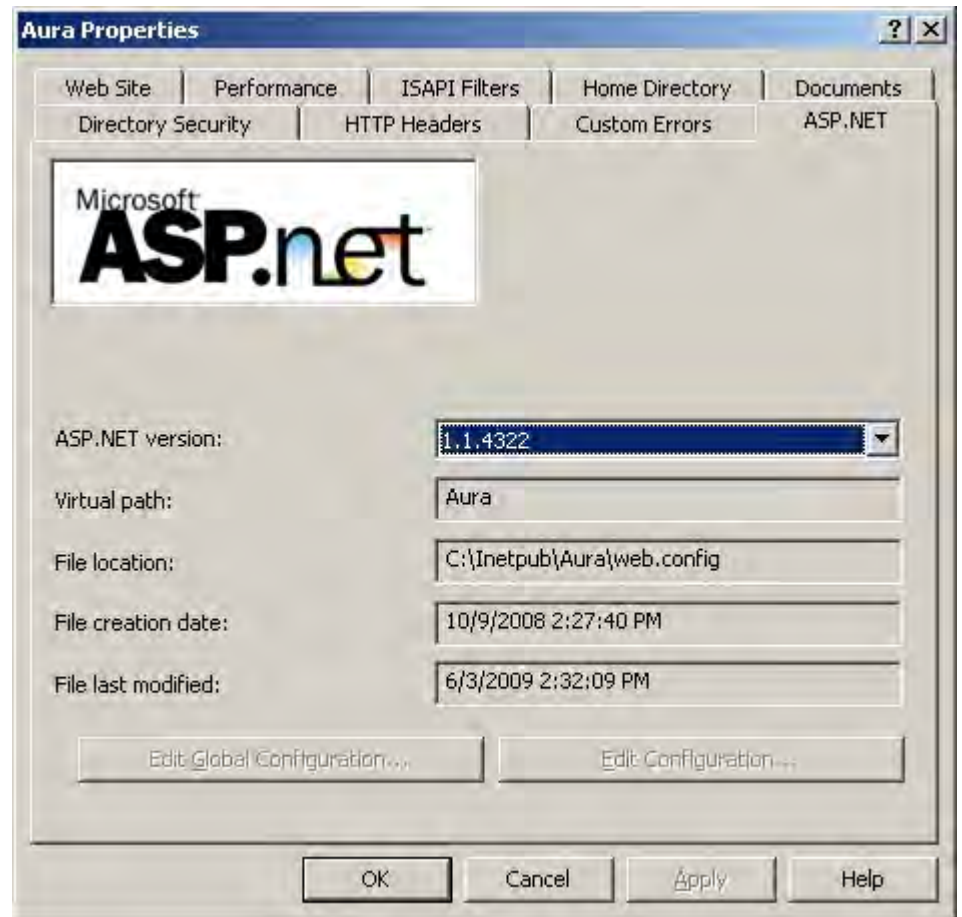
Click **OK** a number of times to exit the open dialog windows.

#### **Aura installation 5 - IIS / ASP.NET 1.1.4322**

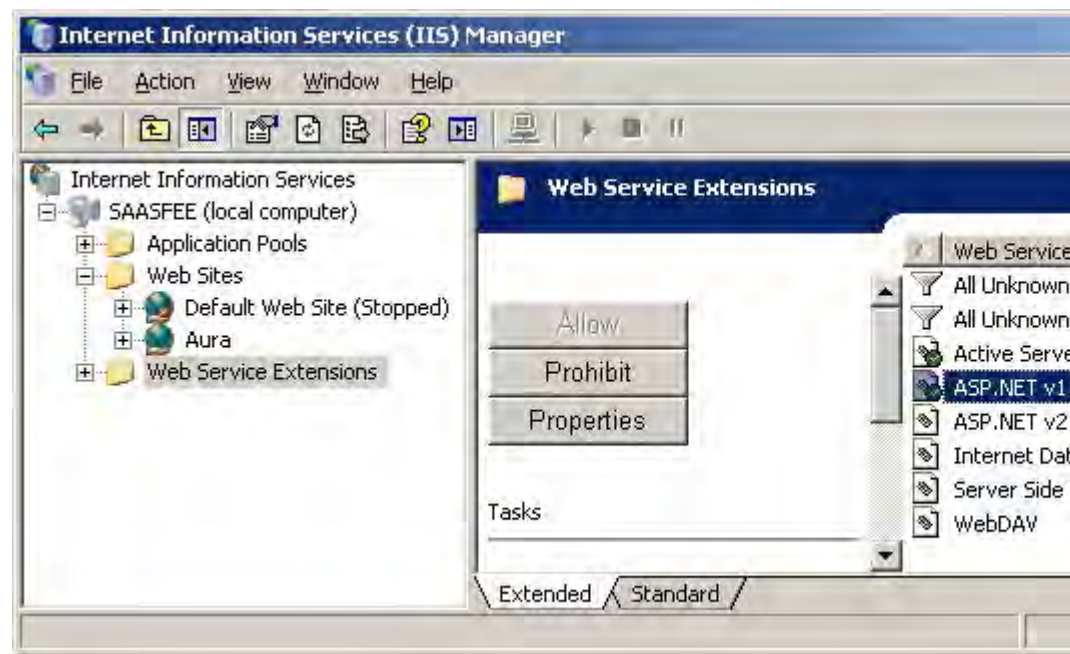
It is required to have ASP.NET installed as part of the IIS service on the computer on which the **UMRA-Aura-WebService** is installed. The installation of ASP.NET on a computer running IIS is different for Windows XP and Windows 2003. ASP.NET is part of the Microsoft's .NET framework. When all updates and service packs of the OS are installed, the most recent version of ASP.NET is by default available for IIS. As of this writing, the most recent version is ASP.NET 2.0. This is not the correct version. Instead ASP.NET 1.1 is required.

**For instructions how to enable ASP.NET 1.1.4322 see <http://support.microsoft.com/kb/816782> for more information.**

Once installed and configured, select the Aura web-site and check the properties. The ASP version should look similar as in the following picture:

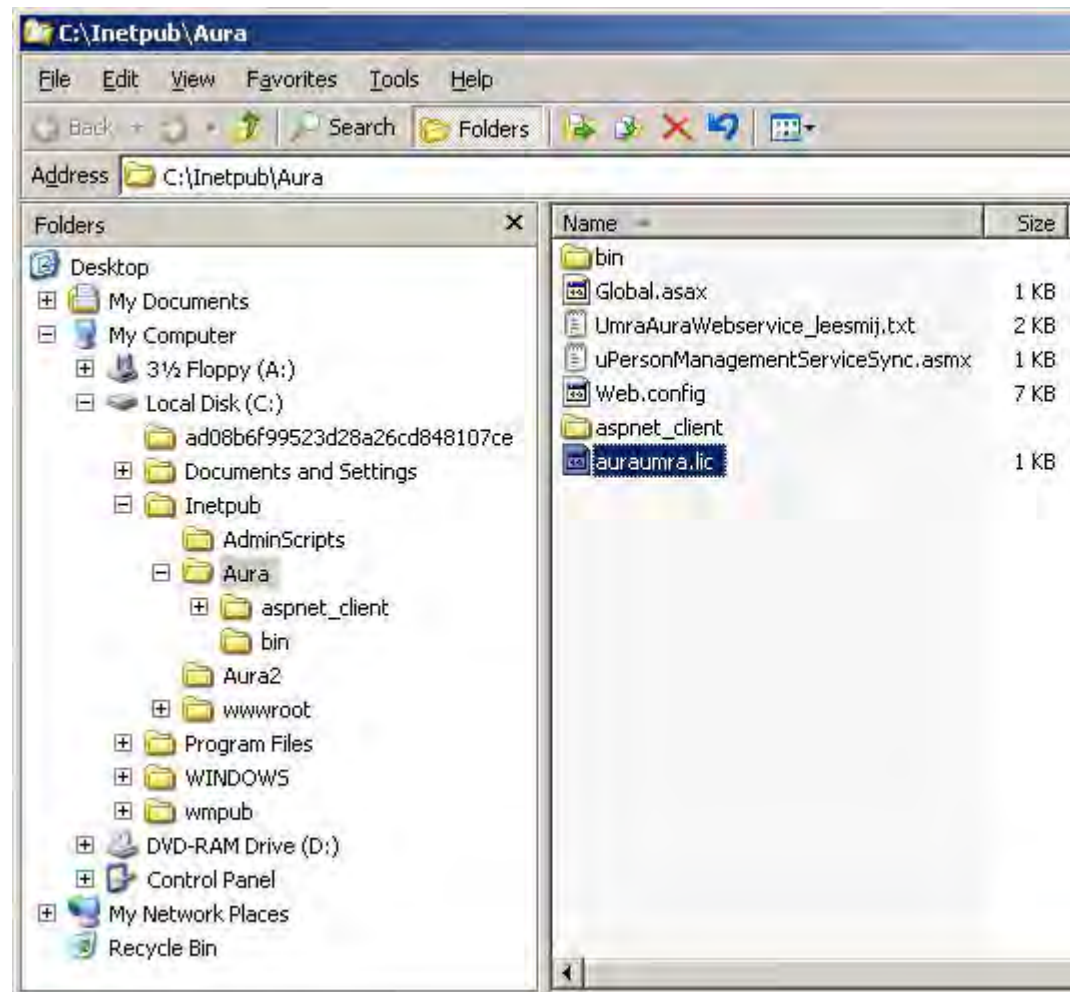


Further, on Windows 2003, the Web Service Extension for ASP.NET v.1.1.4322 must be enabled as shown in the following screenshot:



#### Aura installation 6 - Aura license file

To run properly, a license is required for the UMRA-Aura-WebService. Request a license from the Aura support desk. The license is contained in a file with name **auraumra.lic**. Copy the file to the directory of the web-site of the UMRA-Aura-WebService. The directory of the web-site should now have the following contents:



#### Aura installation 7 - Aura data access

The UMRA-Aura-WebService must be able to access the Aura database server. In case the UMRA-Aura-WebService is running on a server platform, e.g. Windows Server 2003, a client/server license is required and the Aura data is accessed using a host - port combination. In this case, no special steps are required to access the data. If not running a client/server license, the Aura database server is accessed through file sharing. If the standard Aura software and the UMRA-Aura-WebService are running on the same computer, not special steps are required. In all other cases, a share must be created to access the Aura database server.

**To summarize, execute the following steps only if:**



1. **The standard Aura software and the UMRA-Aura-WebService run on different computers and**
2. **Aura is running without using a client/server license.**

To setup the share, perform the following steps on the computer that runs standard Aura software:

1. Locate the **auradb.add** file on computer that runs the Aura standard software;
2. Setup a share for the directory that contains the file. The file is now accessible using UNC path: `\\COMPUTER\Sharename\auradb.add`;
3. Setup the share permissions and grant **Full Control** access rights for the *account used by the UMRA-Aura-WebService*.
4. Modify the NTFS permissions of the directory that contains the the **auradb.add** file. Grant **Modify, Read & Execute, List Folder Contents, Read and Write permissions** for the new account. Propagate the permissions to the directory itself and all files contained in the directory.

#### **Aura installation 8 - Update web.config**

The UMRA-Aura-WebService used configuration file web.config for a number of settings. The file is located in the directory of the UMRA-Aura-WebService web-site. The following settings must be updated:

#### **Data source reference**

Find element `<appSettings>`, child element `<add key="DataSource" ...>` in the file. The value contains the directory of the Aura database server. If a share is used, specify the UNC path. If the standard Aura software is running on the same computer, the full path name can be used. Refer to the comment section in the file for more information.

Examples:

If running the Aura software on computer BERN, and the file auradb.add is located in a directory shared with name Aurawin:

```
<appSettings>
    <add key="DataSource" value="\\BERN\Aurawin\auradb.add" />
    ...
</appSettings>
```

If running on the same computer:

```
<appSettings>
    <add key="DataSource" value="D:\auradb\auradb.add" />
    ...
</appSettings>
```

#### **Debug log file**

It is possible to log the operations and errors of the UMRA-Aura-WebService to a file. The file is specified as follows:

```
<appSettings>
    ...
    <add key="DebugSoapFile" value="C:\AuraLog\debugSOAP.log"
/>
</appSettings>
```

The directory of the file must exist and the account used by the UMRA-Aura-WebService (see further) must have write access to the file. It is recommended to enable the logfile during installation and testing. When the installation is complete, debug logging is disabled by updating the entry in the configuration file:

```
<appSettings>
```

```
...
```

```
<!--add key="DebugSoapFile"
value="C:\AuraLog\debugSOAP.log" /-->

</appSettings>
```

### Identity

When the UMRA-Aura-WebService is running on **Windows XP**, the account that is used to access the Aura database server must be specified. Insert the following element just before the termination element `</system.web>`:

```
<system.web>

...

...

    <identity impersonate="true" userName="domain\username"
password="[password]"/>

</system.web>
```

Example, with username `UmraAuraUser` of domain `DOM_A` and password `[secret3635]`:

```
<system.web>

...

...

    <identity impersonate="true"
userName="DOM_A\UmraAuraUser" password="[secret3635]"/>

</system.web>
```

**Aura installation 9 - Test the web-site**

Before using UMRA to access the connector, it is a good idea to test the web-site first. Before running the test, enable debug logging. See *Aura installation 8 - Update web.config* for more information. On the computer that runs the UMRA-Aura-WebService, start Internet Explorer and connect to file

**`http://localhost/uPersonManagementServiceSync.asmx`**. The result should look like this:



If nothing is shown or an error message shows up, something is wrong with the configuration. Check the UMRA-Aura-WebService debug log file for more information. If all is fine, the log file contains similar information as shown below:

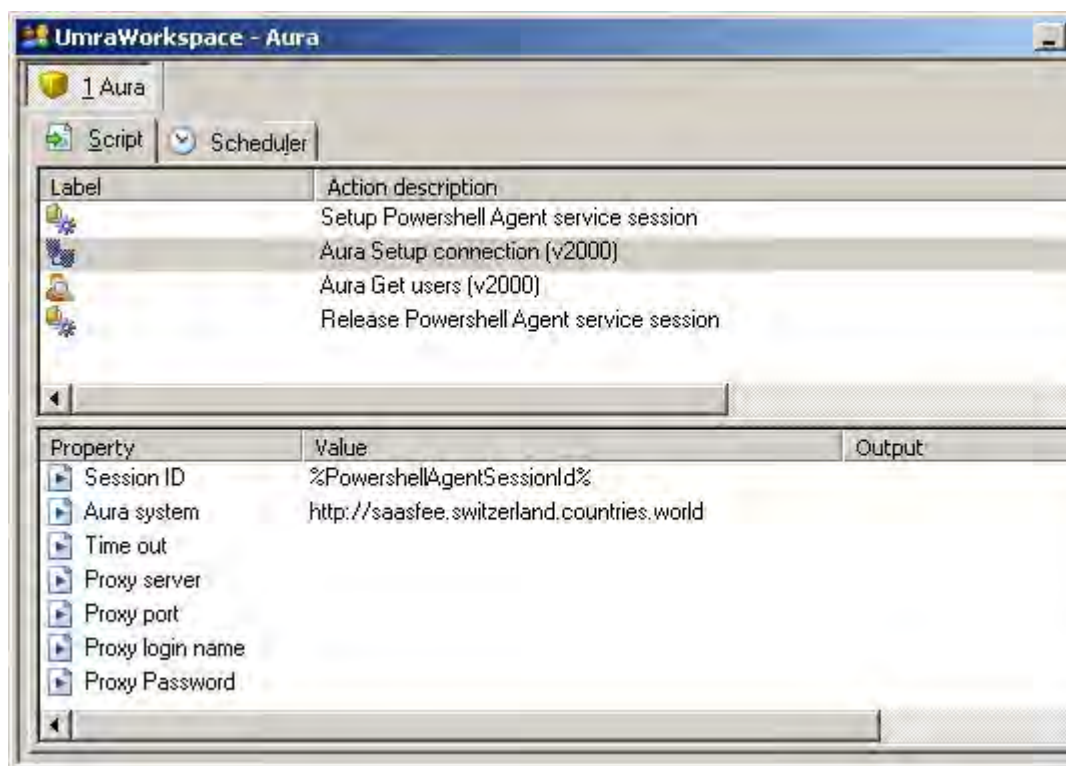
```
Umra Aura-webservice
Versie9.01
Start: 4-6-2009 14:21:0:51
ConnectionString=Data Source =
\\BERN\AuraWin\auradb.add; User ID = AuraNET;
Password = AuraNET!; TableType = CDX; ServerType
= REMOTE|LOCAL; Shared = TRUE; Pooling = FALSE
ServerName=BERN
ServerType=LOCAL
PasnummerIsLenersCode=1
DebugSoapFile=C:\AuraLog\debugSOAP.log
deletePersonAllowed=1
```

```
Licentie gegevens.
Module: Aura-Umra webservice (030)
Versie: 9
Bestandsnaam: C:\Inetpub\Aura\auraumra.lic
Licentiehouder: Aura testlicentie
Licentie is geldig
```

```
Einde umra Aura-webservice: 4-6-2009 14:41:29:681
```

#### **Aura installation 10 - Test the UMRA-Aura-WebService**

When the test of step 9 is completed successfully, UMRA can be used to test the UMRA-Aura-WebService. For this test it is assumed that the UMRA Console, Service and Powershell Agent service are installed. With the UMRA Console application, setup a small automation project as shown below:



When the project is executed, no errors should occur. The Aura debug log file, UMRA Powershell Agent service and UMRA Service log file contain information describing the execution of the project.

### 3.25. SOAP Synchronization template project

This topic describes a general synchronization project that can be used to implement a connector with a particular system.

Creating a PowerShell Initialization Script.

The text <Product> should be replaced some word, describing the name of the system for which the PowerShell session is used. For example 'Google, TeleTOP, etc.

1. Create a new automation script ('<Product>Sync\_Init').
2. a. Add the action 'Check session variable'
  - b. Set the property 'Variable name' to '%G\_<Product>PowerShellSession%'
  - c. Set the output property 'Variable exists flag' to '%<Product>PowerShellSessionExists%'
  - d. Set the on error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to '<Product> Sync init: Error, could not determine if the PowerShell session exists.'
3. a. Add the action 'If-Then-Else'
  - b. Add a if-criteria (Variable Name = '%<Product>PowerShellSessionExists%', Variable type = 'boolean (yes/no, true/false)', Operator = 'equal', Value = 'No')
  - c. Set the 'Then Goto label' to '<Product>\_session\_create'
4. a. Add the action 'Check Powershell Agent service session'
  - b. Set the property 'Session ID' to '%G\_<Product>PowerShellSession%'
  - c. Set the output property 'Variable exists flag' to '%<Product>PowerShellSessionIsValid%'
  - d. Set the on error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to '<Product> Sync init: Error, could not determine if the PowerShell session '%G\_<Product>PowerShellSession%' is valid.'
5. a. Add the action 'If-Then-Else'



b. Add a if-criteria (Variable Name = '%<Product>PowerShellSessionIsValid%', Variable type = 'boolean (yes/no, true/false)', Operator = 'equal', Value = 'Yes')

c. Set the 'Then Goto label' to '<Product>\_session\_exists'

6. a. Add the action 'No operation'

b. Set the label to '<Product>\_session\_create'

7. a. Add the action 'Setup Powershell Agent service session'

b. Set the output property 'Session ID' to '%G\_<Product>PowerShellSession%'

c. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to '<Product> Sync init: Error, could not setup a PowerShell session.'

8. a. <place some product specific actions to initialize the session (such as setup connection) here>

b. Make sure for every! (even a set variable) action the On error is set to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to '<Product> Init: <specific error message here>'

9. a. Add the action 'Set session variable'

b. Set the property 'Variable name' to '%G\_<Product>PowerShellSession%'

c. Set the On error to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to '<Product> Sync init: Error, could not save the <product> PowerShell session to the UMRA session.'

10. a. Add the action 'No operation'

b. Set the label to <Product>\_session\_exists

11. a. <place some product specific actions (after a session is setup) here. For example actions to initialize the user table.>

b. Make sure for every major action the On error is set to: A jump to label 'ERROR', Set variable '%OperationStatus%' to '<Product> Init: Error, <specific error message here>'

12. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

13. a. Add the action 'No operation'

b. Set the label to 'ERROR\_CLEANUP\_POWERSHELL'

14. a. Add the action 'Release Powershell Agent service session.'

b. Set the property 'Session ID' to '%G\_<Product>PowerShellSession%'

15. a. Add the action 'Delete session variable'

b. Set the property 'Variable name' to  
'%G\_<Product>PowerShellSession%'

16. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

17. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

18. a. Add the action 'No operation'

b. Set the label to 'ERROR'

19. <Place the same action as in step 13 here.>

20. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

21. a. Add the action 'No operation'

b. Set the label to 'END'

Save the script.

Creating a PowerShell Cleanup Script.

The text <Product> should be replaced some word, describing the name of the system for wich the PowerShell session is used. For example 'Google, TeleTOP, etc.

1. Create a new automation script ('<Product>Sync\_Cleanup').

2. a. Add the action 'Check session variable'

b. Set the property 'Variable name' to  
'%G\_<Product>PowerShellSession%'

c. Set the output property 'Variable exists flag' to  
'%<Product>PowerShellSessionExists%'

d. Set the on error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to '<Product> Sync cleanup: Error, could not determine if the PowerShell session 'G\_<Product>PowerShellSession' exists.'

3. a. Add the action 'If-Then-Else'

b. Add a if-criteria (Variable Name = '%<Product>PowerShellSessionExists%', Variable type = 'boolean (yes/no, true/false)', Operator = 'equal', Value = 'No')

c. Set the 'Then Goto label' to 'END'

4. a. Add the action 'Release Powershell Agent service session'

b. Set the property 'Session ID' to '%G\_<Product>PowerShellSession%'

5. a. Add the action 'Delete Session Variable'

b. Set the property 'Variable name' to '%G\_<Product>PowerShellSession%'

6. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

7. a. Add the action 'No operation'

b. Set the label to 'ERROR'

8. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

9. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'

10. a. Add the action 'No operation'
  - b. Set the label to 'END'

Creating the Example source script.

1. Create a new automation script. ('ExampleSource\_Init')
2. a. Add the action: 'Generate generic table'
  - b. Set the table type to 'File (text, csv)'
  - c. Specify the example file.
  - d. Check the checkbox 'First line contains headers'
  - e. Uncheck the checkbox 'Insert row number column'
  - f. Specify the variable '%ExampleSourceUsers%' in the 'Variable' tab
  - g. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'Example Source init: Error, could not load the example source data.'
3. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'
4. a. Add the action 'No operation'
  - b. Set the label to 'ERROR'

5. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

6. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

7. a. Add the action 'No operation'

b. Set the label to 'END'

## **4. UMRA Reference Guide**





---

## 4.1. Script action overview

This section contains an overview of all available script actions.

### 4.1.1. User

#### Active Directory

*Script Action: Create user (AD)*

#### Function

Creates a user account in an Active Directory domain. This action is intended to create user accounts in domains and organizational units of Active Directory. In addition to just creating the account itself it also will also configure Active Directory attributes of the account, such as the password and the description of the account.

Some attributes of the user account may specify the usage by the account of other resources in the network. These resources themselves will not be created by this action. If these resources need to be created, this can be done by separate actions that follow this action in the User Management Resource Administrator script. An example of such a property is the Home Directory. When specified in this Create User action, the Home Directory attribute of the user account will be set. The directory itself however is not created. In order to create the directory itself, the script action *File System, Create directory* on page 341 should be performed

This action cannot be used to create accounts outside of Active Directory. In order to create user accounts in a NT4 domain, or to create local user accounts on specific computers, use the action *Script Action: Create User (no AD)* on page 68 instead.

#### Deployment

This action is typically used as core part of a script designed to create users in Active Directory domains, in order to create the account and its attributes itself. In such a script this is usually the first major action invoked. After creating the account, the script usually continues by invoking actions to create home directories, home shares, group memberships, etc.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
Domain	The domain in which to create the user domain account.	%Domain%	Often the domain name is used in many different actions, and is determined and stored in a variable previous to the action ( e.g. %Domain%). The name of the domain can be either in DNS or NETBIOS style. (e.g. Tools4ever.com or TOOLS4EVER). For more information on how to specify the domain/OU/container in which the user account is created, see the <b>Remarks</b> section below.
Organizational Unit-Container	The name of the Active Directory Organizational unit or other container in which to create the account.	Users	Specify the path of the organizational unit (OU) or container relative to the domain. To specify OU's in OU's, use the full path relative to the domain, separated by slashes: OU/ChildOU/GrandChildOU. Examples: students or <b>students/group1</b> . For more information on how to specify the domain/OU/container in which the user account is created, see the <b>Remarks</b> section below.

LDAP container	Optional: The LDAP name of the container in which to create the account.		<p>Optionally specifies name of the Active Directory container in which the user is created directly by means of its LDAP name (Example: CN=users, DC=tools4ever,DC=com Example: OU=Group1, OU=Students, DC=tools4ever, DC=com)</p> <p>This specification can be used instead of the Domain and Organizational Unit-Container properties of this action. If specified, the specified LDAP Container takes precedence, and the Domain And Organization Unit-Container properties are ignored. For more information on how to specify the domain/OU/container in which the user account is created, see the <b>Remarks</b> section below.</p>
----------------	--	--	---

Domain (controller)	Optional: The name of the domain controller or domain used to access the domain.		<p>If this value is not specified, the application creates the account on a domain controller that is determined by Active Directory (serverless binding). If a domain controller is specified, the account is explicitly created on the specified controller (server binding). In both cases, Active Directory itself will replicate the account information to all domain controllers in the forest automatically as required.</p> <p>Depending on the actual User Management Resource Administrator Script used, it may be necessary to specify a domain controller here. If an subsequent script action does an Active Directory query to obtain information of the newly created user, this query may occur before Active Directory has replicated the new information to other Domain Controllers. As a consequence, the query may fail to find the newly created user. When both actions however specify the same domain controller, the newly created user can be found.</p> <p>Often a requery of Active Directory by subsequent actions for the newly created user</p>
------------------------	--	--	--

Name generation algorithm	Specifies the name of the algorithm used to generate user names	Default	<p>The main purpose of the Name Generation algorithm is to create unique names that adhere to your company's syntax requirements.</p> <p>A common implementation of the algorithm will take as input the three variables %FirstName%, %MiddleName% and %LastName%, and generate from these the variables %FullName% and %UserName%. Here %FullName% contains the complete name of the user formatted for display purposes, and %UserName% the name formatted for use as the name of the account. These resulting variables can then be used as input for the other properties of this action</p>
---------------------------	---	---------	--

SAM- Account- Name	The user logon name(Pre- Windows 2000) without the (NETBIOS) Domain name.	%UserName%	<p>This name is required, also in domains that use solely Active Directory domain controllers. This name is usually chosen to be the same as the prefix of the User Principal Name.</p> <p>A SAM-Account-Name cannot be identical to any other user or group name on the domain being administered. It can contain up to 20 uppercase or lowercase characters, except for the following: " / \ [ ] : ;   = , + * &lt; &gt; . A SAM-Account-Name cannot consist solely of periods (.) or spaces.</p> <p>Typically the name contained in %UserName% is generated by the <i>Name generation algorithm</i> on page 121. If the name is found not be unique, the next iteration of the algorithm is tried until unique definite names are generated.</p>
-----------------------	---	------------	---

User-Principal-Name	The User- Principal-Name (UPN) is an Internet style logon name for the user.	%UserName% @Mycompany. com	<p>The UPN is the preferred login name for Active Directory users. Users should be using their UPN to log on to the domain. The UPN has the format account_name@domain.com, where account_name is the UPN prefix and domain.com is the UPN suffix.</p> <p>The UPN Prefix is usually chosen to be the same as the SAM-Account-Name. Typically the name contained in %UserName% is generated by the name generation algorithm.</p>
CommonName	The CommonName is the full name of the user. This name is most commonly used in user interfaces.	%FullName%	Typically the name contained in %FullName% is generated by the name generation algorithm.
DisplayName	This is the Display name attribute of the account. It usually contains the full name of the user.	%FullName%	Typically the name contained in %FullName% is generated by the name generation algorithm.
Given- Name	Optional. The given name corresponds usually with the first name of the user.	%FirstName%	Typically the variable %FirstName% is directly read from the a import file specifying the users to create.

Initials	Optional. The initials of the user. It has a maximum length of six characters.	%MiddleName%	Typically the variable %MiddleName% is directly read from the a import file specifying the users to create.
SurName	Optional. The surname of the user.	%LastName%	Typically the variable %LastName% is directly read from the a import file specifying the users to create.
Password generator	The specification how to generate passwords for the user account		<p>Specifies the method used to generate a password for the user account. These methods vary from simple (easy to remember) passwords to strong passwords. There are several predefined settings available.</p> <p>The resulting password will be stored in a variable. By default it is stored in the variable %Password%. This variable is used as the value for the <b>Password</b> property.</p>



Password	The password for the created account	%Password%	Typically the name contained in the variable %Password% is generated by the <b>Password generator</b> . To create the same password for all users you can specify the password here directly. For example "test1234". You can also read the password from the input file.
Description	A text string, that will be shown in the Description field of the user account in windows. The string can have any length.		

Home directory	The home directory of the user as specified in the "Home folder" setting of the user account	\\%HomeServer%\users\%UserName%	<p>The value can be specified either in the form \\&lt;server name&gt;\&lt;share name&gt;\&lt;rest of path&gt;, or as an local path e.g. G:\UserData\&lt;user name&gt;.</p> <p>Note, This specification does create the home directory itself if it does not exist. In order to create the home directory, specify the action <i>Script Action: Create Directory</i> on page 341 in the User Management Resource Administrator script after this action.</p> <p>Typically the name contained in %UserName% is generated by the <i>name generation algorithm</i> on page 121, and the name contained in \\%HomeServer% is specified previously in the script, or in the import file.</p>
Home directory drive	The drive letter to which the home directory is connected. Specify only the drive letter itself without colon and or backslash		If the drive letter is specified, the Home directory must be specified in the form \\<server name>\<share name>\<rest of path>, and not as a local path.

User profile	The profile path of the user account	\\%HomeServer%\profiles\%UserName%	The value must have the form \\<server name>\<share name>\<rest of path>.
Logon script	Full or relative path to the script file that is executed by Windows when the user logs on	\\%HomeServer%\scripts\%UserName%.bat or %UserName%.bat	If a relative path is specified, this is relative to the default Script directory of Windows.
User must change password at next logon	Specifies whether the user must change the password at the next logon	Yes	Valid specifications are YES and NO. The default value is NO. When set to YES, the <b>User cannot change password</b> property must be set to NO.
User cannot change password	Specifies whether the user is disallowed change the assigned password	No	Valid specifications are YES and NO. The default value is NO. This setting has no effect on members of the administrators group. When set to YES, the <b>User must change password at next logon</b> property must be set to NO.
Password never expires	Specifies whether the password will never expire		Valid specifications are YES and NO. The default value is NO. This setting overrides the <b>Maximum Password Age</b> setting in the password policy for the domain/computer.

Store password using reversible encryption	Specifies whether the password will be stored using reversible encryption	No	Allows a user to log on to a Windows network from Apple computers. If a user is not logging on from an Apple computer, this option should not be used.
Account Disabled	Specifies whether the account should be create in the disabled state	No	Valid specifications are YES and NO. The default value is NO
Smart card is required for interactive logon.	Specifies whether a smart card is required	No	Requires that the user possesses a smart card to log on to the network interactively. The users must also have a smart card reader attached to their computer and a valid personal identification number (PIN) for the smart card. When this option is selected, the password for the user account is automatically set to a random and complex value and the Password never expires account option is set.

Account is trusted for delegation	Specifies whether the account is trusted for delegation	No	Allows a service running under this account to perform operations on behalf of other user accounts on the network. A service running under a user account (otherwise known as a service account) that is trusted for delegation can impersonate a client to gain access to resources on the computer
Account is sensitive and cannot be delegated	Specified that the account cannot be delegated.	No	Allows control over a user account, such as a for guest or temporary account. This option can be user if this account cannot be assigned for delegation by another account
Use Des encryption types for this account	Provides support for Data Encryption Standard (DES)	No	The Default value is NO
Do not require Kerberos preauthentication	Provides support for alternative implementations of the Kerberos protocol	No	The Default value is NO.
Computer account	This is a computer account for a MS Windows NT Workstation/Windows 2000 Professional or Windows NT Server/Windows 2000 Server that is a member of this domain. Default value: 'No'.	No	Specify Yes if the account is computer workstation account.

Account Expiration	Specifies the date after which the account is expired		If not specified, the account will never expire.
Logon hours	The hours the user account can log on to the domain. By default, domain logon is allowed 24 hours a day, 7 days a week.		The value is specified as a text of 42 hexadecimal characters, representing all the hours of a week. The hours of each day are represented by 6 characters.
Workstations	A list of workstation names, separated by ";", on which the user is allowed to logon.		If specified, the user is only allowed to logon when seated at one of the computers (workstation or server) listed. A maximum of 8 computer (workstation or server) names can be specified.  If not specified, such an explicit restriction does not apply.
General - Office	The user's office location This is the person's office location, including the building and office address or number.		
General - TelephoneNumber	The user's phone number		

General - E-mail	The user's e- mail address. The e-mail address appears with the universal principal name suffix (for example, someone@microsoft.com).		
General - Web-Page	The user's home page URL, either on the Internet or in the local intranet site.		
Address - Street	The user's street address		
Address - P.O. Box	The user's post office box number		
Address - City	The city where the user is located		
Address - State/province	The state or province where the user is located		
Address - Zip/Postal Code	The zip or postal code applicable for the user		
Address - Country/region	The user's country or region		The country can be either explicitly chosen from a drop down list, or be specified as text. In the latter case it can also be read from a variable, for instance created by a column from the list of users.
Telephones - Home	The user's home telephone number		

Telephones - Pager	The user's page number		
Telephones - Mobil	The user's mobil telephone number		
Telephones - Fax	The user's fax number		
Telephones - IP phone	The users IP telephone number		
Telephones - Notes	Descriptive information and any comments for this user.		
Organization - Title	The user's title		
Organization - Department	The user's department		
Organization - Company	The users's company		

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage. However, it may be that the actual value of a specific property is required for an successive action in the User Management Resource Administrator script. To facilitate this need, any property can be explicitly configured to be saved in a variable when the action has been performed. For example, when the password of a user is created with the password generator, the resulting password value may be stored in a variable, so it can be exported to a file by an other action in the script.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.



Property	Description	Default variable name	Remarks
SAM-Account-Name	The user logon name(Pre-Windows 2000) without the (NETBIOS) Domain name, that was used to create the account	%UserName%	If more names have been tried as a consequence of the user name generation algorithm, this contains the last name tried.
Common name	The CommonName is the full name of the user. This name is most commonly used in user interfaces.	%FullName%	If more names have been tried as a consequence of the user name generation algorithm, this contains the last name tried.
Password	The password for the new account	%Password%	
User Object Distinguished name	The Object Distinguished name of the just created account	%UserODN%	This name is often used to uniquely identify the User object in AD, and can be used as input in several subsequent actions, usually specified as "LDAP://%UserODN%"

User Security Identifier (SID)	After execution of the action, this property will contain the security Identifier (SID) of the new account. This is an output-only property	%UserSid%	<p>The User-Security Identifier (SID) is created by the Active directory automatically when the user is created. The SID is used when setting permissions, for instance on home directories. The Create User (AD) action copies this value to this property, so it can be stored in a variable for later usage.</p> <p>By default it is stored in the variable %UserSid%. This can then be used later in subsequent actions, for example when permissions for this account must be specified on files and directories.</p>
User object	This Internal application object representing the just created account.	%UserObject%	<p>The User Object is main purpose is to ease subsequent operations on the same account by actions that follow in the script. For several actions this object can be used as input to specify the account the actions work on.</p>

#### Remarks

##### *Domain / OU / Container / LDAP -specification*

User Management Resource Administrator supports several methods to specify the entity (domain, OU or container) in which the user account will be created. These methods differ in the way the property values are specified. The properties involved are: Domain, Organizational Unit-Container, LDAP container. Depending on your network environment and input data, you should choose the method that fits best:

Properties specified	Properties not specified	Example	Description
Domain Organizational Unit-Container	LDAP container	Domain: TOOLS4EVER or tools4ever.com Organizational Unit- Container: STUDENTS/GROUP1	This is most easy method to create user accounts in OU's. To create the account, User Management Resource Administrator will automatically compose the LDAP name of the container to create the user account.
Domain	LDAP container Organizational Unit-Container	TOOLS4EVER or tools4ever.com	Use this method only, to create user accounts in the domain root. No OU is involved.
LDAP container	Domain Organizational Unit-Container	OU=Group1, OU=Students, DC=tools4ever, DC=com	Use this method if you want to specify the OU directory using the LDAP format. If this property is specified, the Domain and Organizational Unit-Container properties are ignored.

*Script Action: Create contact (AD)*

### Function

Creates an Active Directory contact. A contact is an active directory object which contains contact information.

### Deployment

This action is typically used as a part of a script designed to create contacts in Active Directory domains. Contacts are most often used to make communication between different active directories possible. When you create a contact, the contact can not login on the network.

The setting of the contact can be used by users of the network to contact other users or entities that are not connected to the network.

#### Properties

Property Name	Description	Typical setting	Remarks
Domain	The domain in which to create the user domain account.	%Domain%	Often the domain name is used in many different actions, and is determined and stored in a variable previous to the action ( e.g. %Domain%). The name of the domain can be either in DNS or NETBIOS style. (e.g. Tools4ever.com or TOOLS4EVER). For more information on how to specify the domain/OU/container in which the user account is created, see the Remarks section below.

Organizational Unit- Container	The name of the Active Directory Organizational unit or other container in which to create the account.	Users	Specify the path of the organizational unit (OU) or container relative to the domain. To specify OU's in OU's, use the full path relative to the domain, separated by slashes: OU/ChildOU/GrandChildOU. Examples: students or students/group1. For more information on how to specify the domain/OU/container in which the user account is created, see the Remarks section below.
--------------------------------	---	-------	--

LDAP container	Optional: The LDAP name of the container in which to create the account.		<p>Optionally specifies name of the Active Directory container in which the user is created directly by means of its LDAP name (Example: CN=users, DC=tools4ever,DC=com Example: OU=Group1, OU=Students, DC=tools4ever, DC=com)</p> <p>This specification can be used instead of the Domain and Organizational Unit-Container properties of this action. If specified, the specified LDAP Container takes precedence, and the Domain And Organization Unit-Container properties are ignored. For more information on how to specify the domain/OU/container in which the user account is created, see the Remarks section below.</p>
----------------	--	--	--

Domain (controller)	Optional: The name of the domain controller or domain used to access the domain.		<p>If this value is not specified, the application creates the account on a domain controller that is determined by Active Directory (serverless binding). If a domain controller is specified, the account is explicitly created on the specified controller (server binding). In both cases, Active Directory itself will replicate the account information to all domain controllers in the forest automatically as required.</p> <p>Depending on the actual User Management Resource Administrator Script used, it may be necessary to specify a domain controller here. If an subsequent script action does an Active Directory query to obtain information of the newly created user, this query may occur before Active Directory has replicated the new information to other Domain Controllers. As a consequence, the query may fail to find the newly created user. When both actions however specify the same domain controller, the newly created user can be found.</p> <p>Often a requery of Active Directory by subsequent actions for</p>
------------------------	--	--	---

Name generation algorithm	Specifies the name of the algorithm used to generate user names	Default	<p>The main purpose of the Name Generation algorithm is to create unique names that adhere to your company's syntax requirements.</p> <p>A common implementation of the algorithm will take as input the three variables %FirstName%, %MiddleName% and %LastName%, and generate from these the variables %FullName% and %UserName%. Here %FullName% contains the complete name of the user formatted for display purposes, and %UserName% the name formatted for use as the name of the account. These resulting variables can then be used as input for the other properties of this action</p> <p>For a thorough discussion, please see <i>Name Generation</i> on page 121.</p>
CommonName	The CommonName is the full name of the user. This name is most commonly used in user interfaces.	%FullName%	Typically the name contained in %FullName% is generated by the name generation algorithm.



DisplayName	This is the Display name attribute of the account. It usually contains the full name of the user.	%FullName%	Typically the name contained in %FullName% is generated by the name generation algorithm.
Given-Name	Optional. The given name corresponds usually with the first name of the user.	%FirstName%	Typically the variable %FirstName% is directly read from the a import file specifying the users to create.
Initials	Optional. The initials of the user. It has a maximum length of six characters.	%MiddleName%	Typically the variable %MiddleName% is directly read from the a import file specifying the users to create.
SurName	Optional. The surname of the user.	%LastName%	Typically the variable %LastName% is directly read from the a import file specifying the users to create.
Description	A text string, that will be shown in the Description field of the user account in windows. The string can have any length.		
General - Office	The users's office location This is the person's office location, including the building and office address or number.		
General - TelephoneNumber	The user's phone number		

General - E-mail	The user's e-mail address. The e-mail address appears with the universal principal name suffix (for example, someone@microsoft.com).		
General - Web-Page	The user's home page URL, either on the Internet or in the local intranet site.		
Address - Street	The user's street address		
Address - P.O. Box	The user's post office box number		
Address - City	The city where the user is located		
Address - State/province	The state or province where the user is located		
Address - Zip/Postal Code	The zip or postal code applicable for the user		
Address - Country/region	The user's country or region		The country can be either explicitly chosen from a drop down list, or be specified as text. In the latter case it can also be read from a variable, for instance created by a column from the list of users.
Telephones - Home	The user's home telephone number		
Telephones - Pager	The user's pager number		

Telephones - Mobil	The user's mobil telephone number		
Telephones - Fax	The user's fax number		
Telephones - IP phone	The users IP telephone number		
Telephones - Notes	Descriptive information and any comments for this user.		
Organization - Title	The user's title		
Organization - Department	The user's department		
Organization - Company	The users's company		

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage. However, it may be that the actual value of a specific property is required for an successive action in the User Management Resource Administrator script. To facilitate this need, any property can be explicitly configured to be saved in a variable when the action has been performed. For example, when the password of a user is created with the password generator, the resulting password value may be stored in a variable, so it can be exported to a file by an other action in the script.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property Name	Description	Default variable name	Remarks
Common name	The CommonName is the full name of the user. This name is most commonly used in user interfaces.	%FullName%	If more names have been tried as a consequence of the user name generation algorithm, this contains the last name tried.
Contact object	This Internal application object representing the just created contact object.	%ContactObject%	The Contact Object is main purpose is to ease subsequent operations on the same object by actions that follow in the script. For several actions this object can be used as input to specify the object the actions work on.

**Remarks****Domain / OU / Container / LDAP -specification**

User Management Resource Administrator supports several methods to specify the entity (domain, OU or container) in which the user account will be created. These methods differ in the way the property values are specified. The properties involved are: Domain, Organizational Unit-Container, LDAP container. Depending on your network environment and input data, you should choose the method that fits best:

Properties specified	Properties not specified	Example	Description
Domain Organizational Unit- Container	LDAP container	Domain: TOOLS4EVER or tools4ever.com Organizational Unit- Container: STUDENTS/GROUP1	This is the easiest method to create user accounts in OUs. To create the account, User Management Resource Administrator will automatically compose the LDAP name of the container to create the user account.
Domain	LDAP container Organizational Unit-Container	TOOLS4EVER or tools4ever.com	Use this method only, to create user accounts in the domain root. No OU is involved.
LDAP container	Domain Organizational Unit-Container	OU=Group1, OU=Students, DC=tools4ever, DC=com	Use this method if you want to specify the OU directory using the LDAP format. If this property is specified, the Domain and Organizational Unit-Container properties are ignored.

**Related information:**

*UMRA Basics* on page 3

*Script Action: Get user (AD)*

**Function**

Accesses a user account in Active Directory. The action is always used in combination with other subsequent actions. Once the user is found, an internal data structure representing the user account is setup. This structure is stored in a variable (%UserObject%) that can be used by other actions. The action supports several methods to find the user.

## Deployment

This action is typically used in a script that is used to manage, edit or delete existing user accounts. A number of actions are available to manage user accounts. Most of these actions require a input variable (%UserObject%) that holds the user account. When this action is executed successfully, the subsequent actions in the script have access to the user account using this variable.

You have three options to identify the user account.

1. **LDAP name:** The user account is identified by its full LDAP name. Example: cn=John Williams, ou=Schools, dc=Tools4ever, dc=Com. You only need to specify the property LDAP name to identify the user account. Optionally you can specify a domain controller. The user account is always searched for using LDAP. By specifying the name of a domain controller, the program directly binds to the domain controller instead of a domain controller chosen by Active Directory.
2. **Domain, Organizational Unit-Container, FullName:** From these components, User Management Resource Administrator will compose the LDAP name. If necessary, the components are converted to a suitable format. If the FullName is specified but no **Organizational Unit-Container** is specified, User Management Resource Administrator will not be able to find the user account. Optionally you can specify a domain controller. The user account is always searched for using LDAP. By specifying the name of a domain controller, the program directly binds to the domain controller instead of a domain controller chosen by Active Directory.
3. **Domain, Username:** The user account is specified using the NT-style format Domain/UserName. User Management Resource Administrator will convert the name to the full LDAP name. This method requires most resources but does not need the organizational unit to be specified.

If none of these options can be used, you can use the *Script Action: Search object (AD)* on page 146 to search for the user account. The result of the **Search object** action is the LDAP name of the user account that can be used for option 1.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
Domain	The name of the domain (DNS or NETBIOS style, e.g. tools4ever.com or TOOLS4EVER) of the user account. The user account is specified using LDAP. To specify the user account, you have three options. 1: LDAP name (available from network tree browse operations), 2: Domain + Organizational Unit-Container + FullName (the LDAP name is composed from the individual components), 3: Domain + Username (NT-style, LDAP name is searched for). For each option, you need to specify the corresponding properties.	%Domain%	See <b>Deployment</b> section.
Organizational Unit-Container	The name of the Organizational Unit-Container of the user account (example: Students or Students\\Group1). The user account is specified using LDAP. To specify the user account, you have three options. 1: LDAP name (available from network tree browse operations), 2: Domain + Organizational Unit-Container + FullName (the LDAP name is composed from the individual components), 3: Domain + Username (NT-style, LDAP name is searched for). For each option, you need to specify the corresponding properties.		See <b>Deployment</b> section.

Full name	The full name, more precisely known as the common name of the user account in the Organizational Unit-Container - Domain (example: John Williams). The user account is specified using LDAP. To specify the user account, you have three options. 1: LDAP name (available from network tree browse operations), 2: Domain + Organizational Unit-Container + FullName (the LDAP name is composed from the individual components), 3: Domain + Username (NT- style, LDAP name is searched for). For each option, you need to specify the corresponding properties.	%FullName%	See <b>Deployment</b> section.
Username	The pre-Windows 2000 logon name of the user account (example: JWilliams). The user account is specified using LDAP. To specify the user account, you have three options. 1: LDAP name (available from network tree browse operations), 2: Domain + Organizational Unit-Container + FullName (the LDAP name is composed from the individual components), 3: Domain + Username (NT-style, LDAP name is searched for). For each option, you need to specify the corresponding properties.	%UserName%	See <b>Deployment</b> section.



LDAP name	The full LDAP name of the user account. (example: cn=John Williams, ou=Schools, dc=Tools4ever, dc=Com). If this value is specified, it takes precedence and the values for the properties 'Domain', 'Organizational Unit-Container', 'Full name' and 'Username' are ignored and do not have to be specified.		See <b>Deployment</b> section.
Domain controller	Optional: The name of the domain controller, used to access to the domain, container or organizational unit where the account exists. This property can be used for any of the methods used to specify the user account. If this value is not specified, Active Directory chooses one automatically (serverless binding).		See <b>Deployment</b> section.
User Object	An internal data structure representing the user account. This property is an 'output only' property and is generated automatically when the user is found in Active Directory. This property can be used in other script actions, for instance to create an Exchange mailbox, setup group memberships or modify user attributes.	No input value can be specified. Always specify an output variable, for example %UserObject%	

User Security Identifier (SID)	The security identifier (SID) of the user account. This property is an 'output only' property and can be determined when the user is found in Active Directory. The 'User Security Identifier (SID)' is created by the Active Directory automatically when the user account was created. The SID is used when setting permissions, for instance on home directories, Exchange mailboxes etc. The SID is stored by default in the variable %UserSid%.	No input value can be specified.	Specify an output variable value if the SID is needed in subsequent actions.
Globally Unique Identifier (GUID)	The globally unique identifier (GUID) of the user account. This property is an 'output only' property and can be determined when the user is found in Active Directory. The GUID is stored by default in the variable %UserGuid%.	No input value can be specified.	Specify an output variable value if the GUID is needed in subsequent actions.
User account display name	The display name for the user account as found in Active Directory	No Input value can be specified	Specify an output variable if the display name is needed in subsequent actions.

#### Remarks

Each of the properties **Full name**, **Username** and **LDAP name** can be specified as output variables, even if the user account is determined by other than the output properties.

#### Related information:

*UMRA Basics* on page 3

*Script Action: Edit user (AD)*

### Function

Edits an existing user account in Active Directory. The account is identified by a variable containing the User Object. Use the *Script Action: Get user (AD)* on page 31 to find the user first. For the user account, all regular attributes can be changes and/or reset.

### Deployment

This action is typically used as one of the main actions to manage existing user accounts in Active Directory. You can use this action for a single change, for instance resetting the password of an account or multiple changes like home directory, profile directory and Active Directory attributes. To change the common name (full name) of a user account, you should use the *Script Action: Move - rename user (AD)* on page 63 instead.

For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

The **Edit user** action contains a large number of properties. As described above, the **User Object property** is used to identify the user account. Further all the properties are initially not specified. This means that the corresponding Active Directory attributes of the user account are not changed when the action is executed. So only when a property is specified, the attribute is updated in Active Directory.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
User Object	An data structure representing the user account. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	See Deployment section.
SAM- Account- Name	The user logon name (pre- Windows 2000) without the (NETBIOS) domain name. In most cases the SAM- Account- Name is equal to the prefix of the User- Principal- Name and specified by the general %UserName% name variable. The name must be unique within the domain.		Specify the path of the organizational unit (OU) or container relative to the domain. To specify OU's in OU's, use the full path relative to the domain, separated by slashes: OU/ChildOU/GrandChildOU. Examples: <b>students</b> or <b>students/group1</b> . For more information on how to specify the domain/OU/container in which the user account is created, see the <b>Remarks</b> section below.

User-Principal-Name	<p>The User-Principal-Name (UPN) is an Internet-style login name for the user. The UPN is the preferred logon name for Active Directory users. Users should be using their UPNs to log on to the domain. The UPN has the format 'account_name@domain.com', where 'account_name' is the UPN-prefix and 'domain.com' is the upn-suffix. In most cases the User-Principal-Name prefix is specified by the general user name variable.</p>		<p>The UPN is the preferred login name for Active Directory users. Users should be using their UPN to log on to the domain. The UPN has the format account_name@domain.com, where account_name is the UPN prefix and domain.com is the UPN suffix.</p> <p>The UPN Prefix is usually chosen to be the same as the SAM-Account-Name. Typically the name contained in %UserName% is generated by the name generation algorithm.</p>
DisplayName	<p>This is the Display name attribute of the account. It usually contains the full name of the user.</p>		
Given-Name	<p>The Given-name corresponds with the first name of the user account. The Given-name is an optional attribute of Active Directory user accounts.</p>		
Initials	<p>The 'Initials'-field name corresponds with the middle name of the user account. The 'Initials'-field is an optional attribute of Active Directory user accounts.</p>		

SurName	The 'Surname' corresponds with the last name of the user account. The 'Surname' is an optional attribute of Active Directory user accounts.		
Password generator	The specification how to generate passwords for the user account		<p>Specifies the method used to generate a password for the user account. These methods vary from simple (easy to remember) passwords to strong passwords. There are several predefined settings available.</p> <p>The resulting password will be stored in a variable. By default it is stored in the variable %Password%. This variable must be specified as the value for the <b>Password</b> property.</p>
Password	The password of the user account.		<p>Typically the name contained in the variable %Password% is generated by the <b>Password generator</b>. To create the same password for all users you can specify the password here directly. For example "test1234". You can also read the password from the input file.</p>

Description	A user comment. The field can contain a text of any length.		
Home directory	The path of the home directory of the user account. Note that the home directory is not moved or created by this action. Instead, the home directory specification in the Active Directory is updated. You can move the home directory, by adding the actions 'Copy directory' and 'Delete directory' to the script.		<p>The value can be specified either in the form \\&lt;server name&gt;\&lt;share name&gt;\&lt;rest of path&gt;, or as an local path e.g. G:\UserData\&lt;user name&gt;.</p> <p>Note, This specification does create the home directory itself if it does not exist. In order to create the home directory, specify the <i>Script Action: Create Directory</i> on page 341 in the User Management Resource Administrator script after this action.</p>
Home directory drive	The drive letter to which the home directory is connected. Specify only the drive letter itself without colon and or backslash.		<p>If the drive letter is specified, the Home directory must be specified in the form \\&lt;server name&gt;\&lt;share name&gt;\&lt;rest of path&gt;, and not as a local path.</p>

User profile	A path to the user's profile. Note that this specification does not create the profile directory. Instead, it specifies the profile's path in the SAM user account database. You can create the profile directory, by adding the action 'Create Directory' to the script.		The value must have the form \\<server name>\<share name>\<rest of path>.
Logon script	The path for the user's logon script file. The script file can be a .CMD file, an .EXE file, or a .BAT file.		
User must change password at next logon	The password is expired. Use this property to force the user to change the password at the next logon. Note that the user can logon using the current password.		When set to <b>Yes</b> the <b>User cannot change password</b> property must be set to <b>No</b> .
User cannot change password	The user cannot change password. When the user cannot change the password, only the administrator can change the password.		Valid specifications are <b>Yes</b> and <b>No</b> . This setting has no effect on members of the administrators group. When set to <b>Yes</b> , the <b>User must change password at next logon</b> property must be set to <b>No</b> .



Password never expires	The password should never expire on the account.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . This setting overrides the <b>Maximum Password Age</b> setting in the password policy for the domain/computer.
Store password using reversible encryption	An password specific option. If you have users logging on to your Windows 2000 network from Apple computers, select this option for those user accounts.		Allows a user to log on to a Windows network from Apple computers. If a user is not logging on from an Apple computer, this option should not be used.
Account disabled	The user's account is disabled. If an user account is disabled, the account does exist but cannot be used to logon to the network.		
Smart card is required for interactive logon.	Specifies whether a smart card is required		Requires that the user possesses a smart card to log on to the network interactively. The users must also have a smart card reader attached to their computer and a valid personal identification number (PIN) for the smart card. When this option is selected, the password for the user account is automatically set to a random and complex value and the Password never expires account option is set.

Account is trusted for delegation	Specifies whether the account is trusted for delegation		Allows a service running under this account to perform operations on behalf of other user accounts on the network. A service running under a user account (otherwise known as a service account) that is trusted for delegation can impersonate a client to gain access to resources on the computer
Account is sensitive and cannot be delegated	Specifies that the account cannot be delegated.		Allows control over a user account, such as a for guest or temporary account. This option can be user if this account cannot be assigned for delegation by another account
Use DES encryption types for this account	Provides support for Data Encryption Standard (DES)		
Do not require Kerberos preauthentication	Provides support for alternative implementations of the Kerberos protocol		
Account expiration	Specifies the date after which the account is expired		
Logon hours	The hours the user account can log on to the domain. By default, domain logon is allowed 24 hours a day, 7 days a week.		The value is specified as a text of 42 hexadecimal characters, representing all the hours of a week. The hours of each day are represented by 6 characters.

Workstations	A list of workstation names, separated by ",", on which the user is allowed to logon.		If specified, the user is only allowed to logon when seated at one of the computers (workstation or server) listed. A maximum of 8 computer (workstation or server) names can be specified.  If not specified, such an explicit restriction does not apply.
General - Office	The user's office location. This is the person's office location, including the building and office address or number.		
General - Telephone Number	The user's phone number		
General - E-mail	The user's e-mail address. The e-mail address appears with the universal principal name suffix (for example, someone@microsoft.com).		
General - Web-Page	The user's home page URL, either on the Internet or in the local intranet site.		
Address - Street	The user's street address		
Address - P.O. Box	The user's post office box number		

Address - City	The city where the user is located		
Address - State/province	The state or province where the user is located		
Address - Zip/Postal Code	The zip or postal code applicable for the user		
Address - Country/region	The user's country or region		The country can be either explicitly chosen from a drop down list, or be specified as text. In the latter case it can also be read from a variable, for instance created by a column from the list of users.
Telephones - Home	The user's home telephone number		
Telephones - Pager	The user's pager number		
Telephones - Mobil	The user's mobil telephone number		
Telephones - Fax	The user's fax number		
Telephones - IP phone	The users IP telephone number		
Telephones - Notes	Descriptive information and any comments for this user.		
Organization - Title	The user's title		
Organization - Department	The user's department		
Organization - Company	The users's company		

**See also:**

Help on help

*Script Action: Move - rename user (AD)* on page 63

*UMRA Basics* on page 3

*Script Action: Edit user logon*

**Function**

Edits the logon settings of an existing user account . The account is identified by a variable containing the User Object. Use the *Script Action: Get user (AD)* on page 31 to find the user first. For the user account, all regular attributes can be changed and/or reset.

**Deployment**

This action is typically used as one of the main action to manage existing user accounts in Active Directory. You can use this action for a single change, for instance resetting the password of an account or multiple changes like home directory, profile directory and Active Directory attributes. To change the common name (full name) of a user account, you cannot use this action. Use the *Script Action: Move - rename user (AD)* on page 63 instead to do this.

For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the action *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

The **Edit user logon** action contains a large number of properties. As described above, the **User Object** property is used to identify the user account. Other properties are initially not specified. This means that the corresponding Active Directory attributes of the user account are not changed when the action is executed. Only when a property is specified, the attribute is updated in Active Directory.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
User Object	An data structure representing the user account. Use the action <i>Get user (AD)</i> on page 31 to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	See Deployment section.
Username	The SAM account name of the user for which you want to edit the logon settings.		You should only use this option when you are not using the %UserObject% variable. Instead of the %userObject variable an user account can also be identified by the <b>user name</b> and the <b>domain</b> name or the <b>domain controller</b> .
Domain	The domain in which the user account, for which you want to edit the logon settings, is located.		You should only use this option when you want to identify the user account by <b>username</b> and <b>domain</b> name.
Domain controller	The domain controller of the domain in which the user account, for which you want to edit the logon settings, is located.		You should only use this option when you want to identify the user account by <b>username</b> and <b>domain controller</b> .

Password generator	The specification how to generate passwords for the user account		<p>Specifies the method used to generate a password for the user account. These methods vary from simple (easy to remember) passwords to strong passwords. There are several predefined settings available.</p> <p>The resulting password will be stored in a variable. By default it is stored in the variable %Password%. This variable must be specified as the value for the <b>Password</b> property.</p>
Password	The password of the user account.		<p>Typically the name contained in the variable %Password% is generated by the <b>Password generator</b>. To create the same password for all users you can specify the password here directly. For example "test1234". You can also read the password from the input file.</p>
User must change password at next logon	The password is expired. Use this property to force the user to change the password at the next logon. Note that the user can logon using the current password.		<p>When set to <b>Yes</b> the <b>User cannot change password property</b> must be set to <b>No</b>.</p>

User cannot change password	The user cannot change password. When the user cannot change the password, only the administrator can change the password.		Valid specifications are <b>Yes</b> and <b>No</b> . This setting has no effect on members of the administrators group. When set to <b>Yes</b> , the <b>User must change password at next logon</b> property must be set to <b>No</b> .
Password never expires	The password should never expire on the account.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . This setting overrides the <b>Maximum Password Age</b> setting in the password policy for the domain/computer.
Account disabled	The user's account is disabled. If an user account is disabled, the account does exist but cannot be used to logon to the network.		
Unlock the account	Unlock an user account. When an account is locked it is temporarily impossible to log on to the network. An account gets locked when an incorrect password is specified.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . When set to <b>Yes</b> an locked account will be unlocked. This property can only be used when an account is locked.

**See also:**[Help on help](#)



*Script Action: Move - rename user (AD) on page 63*

*UMRA Basics on page 3*

*Script Action: Get user table (locked out/disabled/password)*

### Function

Script action returning locked-out and disabled users. The resulting table is stored in the variable **%UsersTable%**.

### Deployment

This script action will typically be used in a delegation project with multiple forms to obtain a list a locked-out and disabled users. The result is stored in a variable in table format, containing rows and columns. To show these table data in a form, you have to use the generic form table of the **Variable** type. This action requires the use of an initial project.

### Properties

Property Name	Description	Typical setting	Remarks
Active Directory Root	If set to "Yes", a binding will be established to the root of the Active directory for the currently logged on user or service. If set to "Yes", you need to set the LDAP path property to "No".		
LDAP path	The full LDAP name of the organizational unit, container or domain that must be used for the search (e.g. LDAP://OU=Helpdesk,DC=t4edoc,DC=com		If you only want to obtain a list of user objects in a specific OU, then set the property Active Directory root to "No".

Include all users	Includes all user accounts in the search. When set to "Yes", the properties "Include locked out accounts" and "Include disabled accounts" are ignored.		
Include locked out accounts	If set to "Yes", it will include user accounts that are locked out		
Include disabled accounts	If set to "Yes", it will include user accounts that are currently disabled		
User table		Output is stored in %UsersTable %	

For each returned user object in the table %UsersTable%, the following columns are included:

Column	Description
Name	User name
Description	Description to display for an object
Locked out	"Yes" or "No"
Locked out period [hh:mm:ss]	Specifies the length of time a user is locked out after exceeding the maximum number of invalid password attempts.
Disabled	"Yes" or "No"
Password expired	If "Yes", the password has expired. If "No", the password has not expired.
Password expires	The value is either <b>Expired</b> for those accounts for which <b>Password expired</b> is "Yes" or the number of days before the password will expire.
SAM account name	The logon name used to support clients and servers running older versions of the operating system, such as Windows NT 4.0, Windows 95, Windows 98, and LAN Manager.
Object distinguished name	Same as the Distinguished Name for an object.

User account control flags	<p>Flags that control the behaviour of the user account (e.g. user cannot change password, user is currently locked out, no password required, password never expires, user account is disabled, etc.). The values are given in decimals. If these are converted to hexadecimal values, you can verify which flags are set for the user. Some examples:</p> <p>514 - Disabled users</p> <p>512 - Default account type that represents a typical user</p> <p>See the table under UserAccountControl flags for a full overview.</p>
User lockout time	<p>The date and time (UTC) that this account was locked out. This value is stored as a large integer that represents the number of 100 nanosecond intervals since January 1, 1601 (UTC). A value of zero means that the account is not currently locked out.</p>
Password last set time	<p>The date and time that the password for this account was last changed. The resulting value represents the number of 100 nanosecond intervals since 12:00 AM January 1, 1601. The date represented by this number is in Coordinated Universal Time (UTC). It must be adjusted by the time zone bias in the local machine registry to convert to local time.</p>

#### UserAccountControl flags

This attribute value can be zero or a combination of one or more of the following values:

Hexadecimal value	Description
0x00000001	The logon script is executed.
0x00000002	The user account is disabled.
0x00000008	The home directory is required.
0x00000010	The account is currently locked out.
0x00000020	No password is required.
0x00000040	The user cannot change the password.

0x00000080	The user can send an encrypted password.
0x00000200	This is a default account type that represents a typical user.
0x00000800	This is a permit to trust account for a system domain that trusts other domains.
0x00001000	This is a computer account for a computer that is a member of this domain.
0x00002000	This is a computer account for a system backup domain controller that is a member of this domain.
0x00010000	The password for this account will never expire.
0x00020000	This is an MNS logon account.
0x00040000	The user must log on using a smart card.
0x00080000	The service account (user or computer account), under which a service runs, is trusted for Kerberos delegation. Any such service can impersonate a client requesting the service.
0x00100000	The security context of the user will not be delegated to a service even if the service account is set as trusted for Kerberos delegation.
0x00200000	Restrict this principal to use only Data Encryption Standard (DES) encryption types for keys.
0x00400000	This account does not require Kerberos pre-authentication for logon.
0x00800000	The user password has expired. This flag is created by the system using data from the Pwd-Last-Set attribute and the domain policy.

0x01000000	The account is enabled for delegation. This is a security-sensitive setting; accounts with this option enabled should be strictly controlled. This setting enables a service running under the account to assume a client identity and authenticate as that user to other remote servers on the network.
------------	--

If you want to use the content of the variable **%UsersTable%** in a generic table, you need to set up a generic table of the Variable type. In the setup procedure, you can select the column template **User info** which includes the above mentioned columns.

#### See also:

*Script Action: Get user info* on page 101

*UMRA tables* on page 9

*Script Action: Delete user (AD)*

#### Function

Deletes an existing user account from Active Directory. The account is identified by a variable containing the 'User Object'. Use the action *Script Action: Get user (AD)* on page 31 to find the user first.

#### Deployment

This action is typically used to delete one or more user accounts and associated resources from Active Directory. This action, should be the last action. First the user's resources, e.g. group memberships, home- and profile directories should be deleted.

For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the action *Script Action: Get user (AD)* on page 31. The **Get user** script action supports several ways to find the user and fill the variable.

With this action you can not delete local computer accounts and Windows NT4 domain account. Use *Script Action: Delete user (no AD)* on page 86 instead.

#### Properties

Property Name	Description	Typical setting	Remarks
User Object	An data structure representing the user account. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	See Deployment section.

#### See also:

Help on help

*Script Action: Get user (AD)* on page 31

*Script Action: Move - rename user (AD)* on page 63

*Script Action: Delete user (no AD)* on page 86

*UMRA Basics* on page 3

*Script Action: Set user group memberships (AD)*

#### Function

Makes an Active Directory user account a member of specified Active Directory universal, domain global or domain local groups. The groups can be either security or distribution groups.

#### Deployment

This action is typically used in a script that is intended to create new users in Active Directory, after creation of the actual user account with

*Script Action: Create User (AD)* on page 3. It can also be used for modifying existing accounts.

The groups can be specified by two properties using LDAP names (property: **Group names (LDAP)**) and pre-Windows 2000 names (property: **Group names (Pre-W2K name)**). For both properties, the LDAP name is used to add the user account to the group. For property **Group names (Pre-W2K name)** the LDAP name is searched for in Active Directory. If the group names are known in advance and there is no need to use variables in the specification of the group names, it is recommended to use property **Group names (LDAP)** to specify the names of the groups. In case you want to use pre-Windows 2000 names and variables, it is more convenient to use property **Group names (Pre-W2K name)**. This property contains a list with the pre-Windows 2000 names of the groups. The entries of the list can be a single group name or a variable containing one or more group names specified as a text list. When the action is executed, the application will search in Active Directory to find the LDAP name of the group. The method used to access Active Directory is determined by the syntax used to specify the group name:

Syntax	Example	Description
GroupName	Administrators	The Active Directory path of the %UserObject% property is used to access Active Directory.
Domain\GroupName	SEASONS\Administrators	The application accesses Active Directory through the domain: LDAP://Domain
\\Server\GroupName	\\SPRING\Administrators	The application accesses Active Directory by accessing the server: LDAP://Server

Note that for each item of the list a different syntax can be used.

A common scenario to specify a number of groups using variables is as follows:

1. A number of *Set Variable* on page 544 script actions are used to initialize multiple variables, each containing a number of groups: %GroupSetA%, %GroupSetB%, %GroupSetC% etc.

2. The *Map variable* on page 567 script actions copies the content of one of these variables into the resulting variable %GroupSet%. The mapping is somehow determined by the content of the input data.
3. The **Group names (Pre-W2K name)** property contains a single entry: %GroupSet%

The mapping performed in step 2 determines the groups of which the user account becomes a member.

#### Properties

Property Name	Description	Typical setting	Remarks
User Object	Internal application object representing the user account that must be made a member of specified groups.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action, e.g. the <i>Script Action: Create User (AD)</i> on page 3 will by default fill the variable %UserObject% with the User Object of the newly created user.



Group names (LDAP)	The names of the groups of which the user account must become a member. Each group name is specified by 2 text strings: A display name and the LDAP name. The display string has the easy readable format Domain/GroupName, for instance: TOOLS4EVER/Users. The LDAP name is the name of the group in Active Directory. The LDAP name is used by the application to add the user to the group.	LDAP group names specified by means of a special dialog	The property is list with text pairs. Each pair represents a single group. The pair items are the display name and the LDAP name of the group,
Group names (Pre-W2K name)	The names of the groups of which the user account must become a member. Each group name is specified by its pre-Windows 2000 name. This name corresponds with the Windows NT naming style. The application will first search for the full LDAP-name of the group. See the on- line help for more information.	Pre- Windows 2000 group names	The property is a list. The list contains the pre-Windows 2000 names of the groups. The name can be specified using the following syntax: DOMAIN\GroupName, \\SERVER\GroupName, GroupName. See the <b>Deployment</b> section for more information.

### Multiple domain controllers

In a scenario where there are multiple domain controllers in a network, the LDAP group name may have to be specified somewhat differently.

After the LDAP names of the groups have been specified, edit the LDAP names of the groups manually and specify the same server for the binding as the one which was used to create the user. This needs to be done for each group of which the user should become a member.

If this procedure is not followed, an error may occur in a situation where this script action is used for a user which has just been created. This is because the **Set User Group Memberships (AD)** action may be executed on a different domain controller than the one on which the user has just been created. This is inherent to the way in which the script action operates:

1. It retrieves the full LDAP user name from the user object.
2. Then it connects to a given domain controller using the specified AD group.
3. Finally, the script actions tells the AD group to make the user a member of the group.

This is where it can go wrong, since this user may not be known yet on any other domain controller than the one on which it has just been created.

**See also:**

[Help on help](#)

*UMRA Basics* on page 3

*Script Action: Remove user group memberships (AD)*

**Function**

Removes the group memberships of an Active Directory user account. You can filter on local, global, universal, security and distribution groups.

## Deployment

This action is typically used in a script that is intended to manage existing user accounts. With this action you can delete the user accounts from all or various groups of which the account is a member. You can define 2 filters to determine the groups from which the user account is deleted:

Filter 1: local - global - universal groups. For each possible value you can specify if the user account must be deleted from the corresponding groups.

Filter 2: security - distributions groups. For each possible value you can specify if the user account must be deleted from the corresponding groups.

The user account is deleted from a group if both filter criteria are met. Example: If you set the filter properties for global and security to Yes and all other filter properties to No, the user account is deleted from a global security group but not from a global distribution group.

## Properties

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing the user account. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action, for example Script Action: Get user (AD).

Remove from local groups	Remove the user account from local groups	Yes	See <b>Deployment</b> section
Remove from global groups	Remove the user account from global groups (scope: local - global - universal).	Yes	See <b>Deployment</b> section
Remove from universal groups	Remove the user account from universal groups (scope: local - global - universal).	Yes	See <b>Deployment</b> section
Remove from security groups	Remove the user account from security groups (scope: security - distribution).	Yes	See <b>Deployment</b> section
Remove from distribution groups	Remove the user account from distribution groups (scope: security - distribution).	Yes	See <b>Deployment</b> section

**See also:**

*Script Action: Set User Group Memberships (AD)* on page 56

*Script Action: Set User Global Group Memberships* on page 88

Help on help

*UMRA Basics* on page 3

*Script Action: Move - rename (AD)*

### Function

Moves a user account in Active Directory to another OU or container within the same domain. Alternatively, you can also use this action to rename a user account in an organizational unit - container of Active Directory.

### Deployment

This action is typically used in a script that is intended to manage existing user accounts. With this action you can execute 2 operations:

1. **Move user account(s) to other organizational units:** The user account can be moved to another organizational unit in the same domain. When the account is moved, the common name of the user account is not changed by default. The common name is part of the full LDAP name of the user account that uniquely identifies the user account in the organizational unit or container. Hence, the common name must be unique in the organizational unit. If you execute this action and move an account to an OU and a user account with the same common name already exists in the OU, the action will fail. Alternatively, you can rename the account (property **NewName**).
2. **Rename a user account:** With this action you can change the common name of the user account. The common name is part of the full LDAP name of the user account that uniquely identifies the user account in the organizational unit or container. Hence, the common name must be unique in the organizational unit. If the new common name is not unique, the action will fail and an error is generated.

You can also combine the 2 possible operations and both move and rename the user account. When you want to move the user account, you need to specify the destination organizational unit or container of the user account. If you only want to rename the user account, the destination organizational unit or container is not changed for the user account. To specify the destination organizational unit or container you have 2 options:

1. Specify properties **Domain** and **Organizational Unit-Container**:  
When moving user accounts to another organizational unit, you must specify the new name of the OU. If the domain is not changed, you don't need to specify property Domain. If you want to use this option, you don't need to specify the property **OU-Container LDAP name**
2. Specify property **OU-Container LDAP name**: If you use this option, you need to specify the full LDAP name of the destination organizational unit - container. Examples: ou=Schools, dc=Tools4ever, dc=Com, LDAP://ou=Schools, dc=Tools4ever, dc=Com, LDAP://domaincontroller/ou=Schools, dc=Tools4ever, dc=Com. With this option, you don't need to specify the properties **Domain** and **Organizational Unit-Container**.

#### Properties

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing the user account. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action, for example <i>Script Action: Get user (AD)</i> on page 31.

Domain	The name of destination domain (DNS or NETBIOS style, e.g. tools4ever.com or TOOLS4EVER) of the user account. If the domain name is not specified, the application assumes that the account is not moved across domains. When no destination Organizational Unit-Container is specified, the user account is not moved but renamed only.		Specification of this property is required only if you want to move and optionally rename the user account across domains.
Organizational Unit-Container	The name of the destination Organizational Unit-Container of the user account (example: Students or Students/Group1). When this property is not specified, the user account is not moved but renamed only unless the property 'OU-Container LDAP name' is specified.		Specification of this property is required only if you want to move and optionally rename the user account to another organizational unit or container.

OU-Container LDAP name	The full LDAP name of the destination Organizational Unit-Container (example: ou=Schools, dc=Tools4ever, dc=Com). When specified, the properties 'Domain' and 'Organizational Unit-Container' are ignored. When no destination Organizational Unit-Container is specified, the user account is not moved but renamed only.		Specification of this property is required only if you want to move and optionally rename the user account to another organizational unit or container.
Domain controller	Optional: The name of the domain controller, used to access to the domain, container or organizational unit where the account is moved to or where the account exists in case of a rename operation. This property 'helps' User Management Resource Administrator to access Active Directory.		
New name	The new name of the user account. The name is the name that identifies the user account in Active Directory e.g. the 'Common- Name'. If this property is not specified, the account is not renamed. To rename other names of user accounts, use the action 'Edit user (AD)'.		You only need to specify this property if you want to rename the user account, e.g. change the common name.

**See also:**

*Script Action: Move cross-domain (AD) on page 67*



*UMRA Basics on page 3*

*Script Action: Move cross-domain (AD)*

### Function

Moves an existing user object (users and computer accounts) from an OU in one domain to an OU in another domain.

### Deployment

This action is typically used in a script that is intended to manage existing user accounts. When moving a user account to another domain, several restrictions apply:

1. The source and destination domain must be in the same forest of domains.
2. The destination domain must be in native mode.

### Properties

The script action **Move cross domain (AD)** has the following properties:

Property Name	Description	Remarks
Source object	The <b>Source object</b> property is the LDAP name of the object to be moved in the original location (before the move).	<b>Important:</b> In case the source domain has multiple domain controllers, the domain controller with the role of RID master must be used to access the source account. Access to the source account is controlled by specifying a binding string as part of the LDAP name:  LDAP://server_rid_master.mydomain.com on page 63/ CN=<AccountToMove>, OU=<SourceOU>, DC=<mydomain>.DC=com.

Target container	The Target container property is used to specify the full LDAP name of the destination of the object. The container can be an organizational unit, domain or general container (e.g. Users). The container must be specified using a server binding string in DNS format: goldfish.marketing.TheFirm.com. This type of specification enforces the move operation to use Kerberos authentication instead of NTLM.	
New name	New name only has to be specified if the (common) name of the user account changes. If not, it can be left unspecified.	

**See also:**

*Script Action: Move - rename user (AD)* on page 63

*UMRA Basics* on page 3

**non- Active Directory**

*Script Action: Create User (no AD)*

**Function**

Creates a user account in an NT4 domain or on a local computer. This action is intended to create user accounts on NT4 domains. Alternatively it can be used to create user accounts on local computers. In addition to just creating the account itself it also will configure several attributes of the account, such as the password and the description of the account.

Some attributes of the user account may specify the usage by the account of other resources in the network. These resources themselves will not be created by this action. If these resources need to be created,

this can be done by separate actions that follow this action in the User Management Resource Administrator script. An example of such a property is the Home Directory. When specified in this Create User action, the Home Directory attribute of the user account will be set. The directory itself however is not created. In order to create the directory itself, the action *Script Action: Create Directory* on page 341 should be performed.

The action may also be used to create user accounts in the default Users container of Active Directory domains. When this action is used to create domain accounts on Active Directory domains, it will correctly create the account in the Active Directory, but many of the Active Directory properties will have default values. To create Accounts in Active Directory with other than default settings, use the action *Script Action: Create User (AD)* on page 3 instead.

### **Deployment**

This action is typically used as core part of a script designed to create users on NT4 domains or local (non domain controller) computers, in order to create the account itself. In such a script this is usually the first major action invoked. After creating the account, the script usually continues by invoking actions to create home directories, home shares, group memberships, etc.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
Domain	The Domain in which to create the user domain account.	%Domain%	Often the domain name is used in many different actions, and is determined and stored in a variable previous to the action ( e.g. %Domain%).  Alternatively the domain name can be specified directly here. Use the NETBIOS (NT4-style) domain name and not the DNS name of the domain This is usually the same as the first part of the DNS domain name.
Computer	The computer on which the local user account is created		If specified, the domain property is ignored, and the account created is a local account on the specified computer, and not a domain account.

Name generation algorithm	Specifies the name of the algorithm used to generate user names		<p>The main purpose of the Name Generation algorithm is to create unique names that adhere to your company's syntax requirements. A common implementation of the algorithm will take as input the three variables %FirstName%, %MiddleName% and %LastName%, and generate from these the variables %FullName% and %UserName%. Here %FullName% contains the complete name of the user formatted for display purposes, and %UserName% the name formatted for use as NT Account. These resulting variables can then be used as input for the other properties of this action.</p> <p>For a thorough discussion, please see <i>Name Generation</i> on page 121.</p>
---------------------------	---	--	--

Username	The name of the user account	%UserName%	<p>A user name cannot be identical to any other user or group name on the computer being administered. It can contain up to 20 uppercase or lowercase characters, except for the following: " / \ [ ] : ;   = , + * &lt; &gt;</p> <p>A user name cannot consist solely of periods (.) or spaces.</p> <p>Typically the name contained in %UserName% is generated by the name generation algorithm.</p>
Full name	The full name of the user	%FullName%	<p>Typically the name contained in %FullName% is generated by the name generation algorithm.</p>

Password generator	The specification how to generate passwords for the user account		<p>Specifies the method used to generate a password for the user account. These methods vary from simple (easy to remember) passwords to strong passwords. There are several predefined settings available.</p> <p>The resulting password will be stored in a variable. By default it is stored in the variable <code>%Password%</code>. This variable is used as the value for the <b>Password</b> property.</p>
Password	The password for the created account	<code>%Password%</code>	<p>Typically the name contained in the variable <code>%Password%</code> is generated by the <b>Password generator</b>. To create the same password for all users you can specify the password here directly. For example "test1234". You can also read the password from the input file.</p>
Description	A text string, that will be shown in the Description field of the user account in windows. The sting can have any length		

Home directory	The home directory of the user as specified in the "Home folder" setting of the user account	\\%HomeServer%\users\ %UserName%	<p>The value can be specified either in the form \\&lt;server name&gt;\&lt;share name&gt;\&lt;rest of path&gt;, or as an local path e.g. G:\UserData\&lt;user name&gt;.</p> <p>Note, This specification does create the home directory itself if it does not exist. In order to create the home directory, specify the action "Create Directory" in the User Management Resource Administrator script after this action.</p> <p>Typically the name contained in %UserName% is generated by the name generation algorithm, and the name contained in \\%HomeServer% is specified previously in the script, or in the import file.</p>
----------------	--	-------------------------------------	--



Home directory Drive	The drive letter to which the home directory is connected. Specify only the drive letter itself without colon and or backslash		If the drive letter is specified, the Home directory must be specified in the form \\<server name>\<share name>\<rest of path>, and not as a local path.
User profile	The profile path of the user account.	\\%HomeServer%\profiles\%UserName%	The value must have the form \\<server name>\<share name>\<rest of path>.
Logon script	Full or relative path to the script file that is executed by Windows when the user logs on	\\%HomeServer%\scripts\%UserName%.bat or %UserName%.bat	If a relative path is specified, this is relative to the default Script directory of Windows.
User must change password at next logon	Specifies whether the user must change the password at the next logon	Yes	Valid specifications are YES and NO. The default value is NO. When set to YES, the "User cannot change password " property must be set to NO.
User cannot change password	Specifies whether the user is disallowed change the assigned password.	No	Valid specifications are YES and NO. The default value is NO. This setting has no effect on members of the administrators group. When set to YES, the "User must change password at next logon" property must be set to NO.

Password never expires	Specifies whether the password will never expire		Valid specifications are YES and NO. The default value is NO. This setting overrides the "Maximum Password Age" setting in the password policy for the domain/computer.
No password required	Specifies whether it is allowed to specify an empty Password value for the user account.	No	Valid specifications are YES and NO. The default value is NO. Setting this value to YES allows empty passwords to be specified. For security reasons it is strongly advised to set this property to NO. If not specified, the password is required.
Computer account	This is a computer account for a MS Windows NT Workstation/Windows 2000 Professional or Windows NT Server/Windows 2000 Server that is a member of this domain. Default value: 'No'.	No	Specify Yes is the account represents a computer - workstation account.
Account disabled	Specifies whether the account should be create in the disabled state.		Valid specifications are YES and NO. The default value is NO.
Account expiration	Specifies the date after which the account is expired		If not specified, the account will never expire.

Logon hours	The hours the user account can log on to the domain. By default, domain logon is allowed 24 hours a day, 7 days a week.		The value is specified as a text of 42 hexadecimal characters, representing all the hours of a week. The hours of each day are represented by 6 characters.
Workstations	A list of workstation names, separated by ",", on which the user is allowed to logon.		If specified, the user is only allowed to logon when seated at one of the computers (workstation or server) listed. A maximum of 8 computer (workstation or server) names can be specified.  If not specified, such an explicit restriction does not apply.
Special user comment	A text string containing additional comments		This property of an user account is not exposed in the User Manager for Domains on a NT 4 machine, or the local accounts snap-in on windows 2000,XP and 2003 computers, but may be shown for informational purposes in other applications.

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage. However, it may

be that the actual value of a specific property is required for an successive action in the User Management Resource Administrator script. To facilitate this need, any property can be explicitly configured to be saved in a variable when the action has been performed.

For example, when the password of a user is created with the password generator, the resulting password value may be stored in a variable, so it can be exported to a file by an other action in the script.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with a blue arrow in the properties list.

Property	Description	Default variable name	Remarks
User name	The name of the user account	%UserName%	If more names have been tried as a consequence of the user name generation algorithm, this contains the last name tried.
Full name	The full name of the user	%FullName%	If more names have been tried as a consequence of the user name generation algorithm, this contains the last name tried.
Password	The password for the created account	%Password%	

**See also:**

*UMRA Basics* on page 3

*Script Action: Edit user (no AD)*

### Function

Edits an existing user account. All main properties and attributes of the account, including password, full name, home directory settings etc. can be modified with this action.

### Deployment

This action is typically used as one of the main action to manage existing user accounts. The account can be a:

- Windows NT 4 domain account
- Local workstation or member server account
- Active Directory account. For an Active Directory account, you can also use *Script Action: Edit user (AD)* on page 37 to edit the account.

To execute this action you need to specify the properties that identify the user account: **Username** and **Domain** or **Computer**. To edit a user account on an Active Directory workstation, you need to specify the name of the workstation for the Computer property. By default, all properties that effect the user account are not specified, e.g. nothing is changed for the user account. By specifying one or more properties, changes are made.

### Properties

Property Name	Description	Remarks
Domain	The name of the domain of the user account. The domain can be specified using with a DNS or NETBIOS name. If the <b>Computer</b> property is specified, this property is ignored.	To specify the user account, specify properties <b>Username</b> and <b>Domain</b> or <b>Computer</b> .

Computer	The name of the computer that maintains the user account. This computer can be specified with a DNS or NETBIOS name. The computer can be a domain controller of a Windows NT4/2000/2003 domain, a member server of a domain or a workstation. If this property is specified, the 'Domain' property is ignored.	To specify the user account, specify properties Username and Domain or Computer.
Username	The name of the user account. The name equals the SAM account name of the user account.	To specify the user account, specify properties Username and Domain or Computer.
Full name	The full name of the user account. When specified, the current name of the user account is changed into the name specified.	
Password generator	A password can be generated automatically. The 'Password generator' specifies how the password is generated, e.g. password length, password complexity requirements, password output variable etc. When this property is specified the password is generated automatically. The password output variable (default: %Password%) should correspond with the variable generated by the password generator.	
Password	The password of the user account.	
Description	A description associated with the user account. The field can contain a text of any length.	

Home directory	The path of the home directory of the user account. Note that this specification does not create the home directory. Instead, it specifies the home directory in the SAM user account database. You can create the home directory, by adding the action 'Create Directory' to the script.	
Home directory drive	The drive letter assigned to the user's home directory for logon purposes.	
User profile	A path to the user's profile. Note that this specification does not create the profile directory. Instead, it specifies the profile's path in the SAM user account database.	
Logon script	The path for the user's logon script file. The script file can be a .CMD file, an .EXE file, or a .BAT file.	
User must change password at next logon	The password is expired. Use this property to force the user to change the password at the next logon. Note that the user can logon using the current password.	
User cannot change password	The user cannot change password. When the user cannot change the password, only the administrator can change the password.	
Password never expires	The password should never expire on the account.	
No password required	No password is required for the user account.	

Account disabled	The user's account is disabled. If an user account is disabled, the account does exist but cannot be used to logon to the network.	
Account expiration	The time and date when the account expires. The value can be 'Never' or a time and date.	
Logon hours	The hours the user account can log on to the domain. By default, domain logon is allowed 24 hours a day, 7 days a week.	The value is specified as a text of 42 hexadecimal characters, representing all the hours of a week. The hours of each day are represented by 6 characters.
Workstations	Optional: the names of the workstations from which the user can log on (8 maximum), separated by commas.	
Special user comment	A user comment. The field can contain a text of any length.	

**See also:**

*UMRA Basics* on page 3

*Script Action: Create User (no AD)* on page 68

*Script Action: Edit user (AD)* on page 37

*Script Action: Edit user logon*

**Function**

Edits the logon settings of an existing user account . The account is identified by a variable containing the User Object. Use the *Script Action: Get user (AD)* on page 31 to find the user first. For the user account, all regular attributes can be changed and/or reset.

**Deployment**

This action is typically used as one of the main action to manage existing user accounts in Active Directory. You can use this action for a single change, for instance resetting the password of an account or multiple



changes like home directory, profile directory and Active Directory attributes. To change the common name (full name) of a user account, you cannot use this action. Use the *Script Action: Move - rename user (AD)* on page 63 instead to do this.

For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the action *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

The **Edit user logon** action contains a large number of properties. As described above, the **User Object** property is used to identify the user account. Other properties are initially not specified. This means that the corresponding Active Directory attributes of the user account are not changed when the action is executed. Only when a property is specified, the attribute is updated in Active Directory.

#### Properties

Property Name	Description	Typical setting	Remarks
User Object	An data structure representing the user account. Use the action <i>Get user (AD)</i> on page 31 to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	See Deployment section.

Username	The SAM account name of the user for which you want to edit the logon settings.		You should only use this option when you are not using the %UserObject% variable. Instead of the %userObject variable an user account can also be identified by the <b>user name</b> and the <b>domain</b> name or the <b>domain controller</b> .
Domain	The domain in which the user account, for which you want to edit the logon settings, is located.		You should only use this option when you want to identify the user account by <b>username</b> and <b>domain</b> name.
Domain controller	The domain controller of the domain in which the user account, for which you want to edit the logon settings, is located.		You should only use this option when you want to identify the user account by <b>username</b> and <b>domain controller</b> .
Password generator	The specification how to generate passwords for the user account		<p>Specifies the method used to generate a password for the user account. These methods vary from simple (easy to remember) passwords to strong passwords. There are several predefined settings available.</p> <p>The resulting password will be stored in a variable. By default it is stored in the variable %Password%. This variable must be specified as the value for the <b>Password</b> property.</p>

Password	The password of the user account.		Typically the name contained in the variable %Password% is generated by the <b>Password generator</b> . To create the same password for all users you can specify the password here directly. For example "test1234". You can also read the password from the input file.
User must change password at next logon	The password is expired. Use this property to force the user to change the password at the next logon. Note that the user can logon using the current password.		When set to <b>Yes</b> the <b>User cannot change password property</b> must be set to <b>No</b> .
User cannot change password	The user cannot change password. When the user cannot change the password, only the administrator can change the password.		Valid specifications are <b>Yes</b> and <b>No</b> . This setting has no effect on members of the administrators group. When set to <b>Yes</b> , the <b>User must change password at next logon</b> property must be set to <b>No</b> .
Password never expires	The password should never expire on the account.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . This setting overrides the <b>Maximum Password Age</b> setting in the password policy for the domain/computer.

Account disabled	The user's account is disabled. If an user account is disabled, the account does exist but cannot be used to logon to the network.		
Unlock the account	Unlock an user account. When an account is locked it is temporarily impossible to log on to the network. An account gets locked when an incorrect password is specified.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . When set to <b>Yes</b> an locked account will be unlocked. This property can only be used when an account is locked.

**See also:**

Help on help

*Script Action: Move - rename user (AD) on page 63*

*UMRA Basics on page 3*

*Script Action: Delete user (no AD)*

**Function**

Deletes a user account from a NT4 domain or local computer.

**Deployment**

This action is typically used as core part of a script designed to delete user accounts. With this action you can delete user accounts from NT4 domains, member servers and workstations and local computers. You can also delete user accounts from Active Directory domains running

Windows 2003/2000 but for Active Directory it is recommended to use *Script Action: Delete user (AD)* on page 55 instead.

The user account that must be deleted is specified by the name of user account and the domain or computer.

**Properties**

Property Name	Description	Remarks
Domain	The name of the domain from which the account is deleted. The domain can be specified using with a DNS or NETBIOS name. If the 'Computer' property is specified, this property is ignored.	
Computer	The name of the computer from which the account is deleted. This computer can be specified with a DNS or NETBIOS name. The computer can be a domain controller of a Windows NT4/2000/2003 domain, a member server of a domain or a workstation. If this property is specified, the 'Domain' property is ignored.	
Username	The name of the user account that must be deleted.	The name is the SAM account name.

**See also:**

*UMRA Basics* on page 3

*Script Action: Delete user (AD)* on page 55

*Script Action: Setup User Global Group Memberships***Function**

Makes a Active Directory or NT4 user account member of a global group. The global group can be a global group from a Active Directory, or an NT4 domain. In both cases the group is identified by its NT4-style (NETBIOS) name. The user and the groups must be all in the same domain. The groups may be either security groups or distribution groups.

**Deployment**

This action is typically used in a script that is intended to create new users in Active Directory or NT4 domains, after creation of the actual user account with *Script Action: Create User (AD)* on page 3 or *Script Action: Create User (no AD)* on page 68. This action is then used to make the users member of a global group.

**Properties**

Property Name	Description	Typical setting	Remarks
Domain	The NT4 style (NETBIOS) name of the domain that contains the global groups	%Domain%	If a DNS-style domain is given, this is converted to a NT4-style domain name by truncating at the first "." encountered in the name.
Domain controller	Optional: the name of the NT4 style (NETBIOS) name of the domain controller of the domain that contains the groups		If a value for the domain controller is specified, the value entered in the domain property is not used.

Username	The NT4 or Pre-Windows 2000 user logon name of the user that must be added to the groups.		The logon name must exist on the domain or domain controller specified in order for the action to succeed.  Any domain names and or backslashes that are specified in this field are automatically stripped from the user name before setting this property
Global groups	A list of global group names of whom the user is to be made a member.  Multiple names can be specified by using a comma "," as separator.		The groups must exist on the domain or domain controller specified in order for the action to succeed. Any domain names and or backslashes that are specified in this field are automatically stripped from the user name before setting this property. For more information on the specification of groups using variables, see <i>Data specification - Text list</i> on page 624.
Remove from other global groups	Indicates whether or not the user must be removed from all other global groups	No	
Error if already member	When set, no error is generated when the user account is already a member of the global group. Default value: 'No'.	No	

**See also:**

*UMRA Basics* on page 3

*Script Action: Map variable* on page 567

*Script Action: Add account to local group*

**Function**

Adds an existing user or global group account to a local group of a domain, server or workstation.

**Deployment**

This action is typically used in a script that manages user accounts and local group memberships. The action can be used in Active Directory, Windows NT domains or workgroup environment. The account is an existing user or global group account. In case the user account is created in the same script, or the user is searched for in Active Directory the security identifier (SID) of the user account can be used to specify the new local group member.

The target local group is one of the following:

1. Active Directory domain local group. In this case you can also use *Script Action: Set User Group Memberships (AD)* on page 56 to add the account to the local group;
2. Windows NT4 domain local group. The group is a local group of the domain, maintained on the primary and backup domain controllers of the Windows NT4 domain.
3. Member server local group. The server is not a domain controller and either a member server of an Active Directory domain, Windows NT4 domain or a workgroup.
4. Workstation local group. The workstation is either a member server of an Active Directory domain, Windows NT4 domain or a workgroup.

Depending on the type of local group, you must specify the **Local group name** and the **Domain** or **Computer property** to identify the local group to which the new member is added.

The new member is specified by either the name (property: **Member (name)**) or security identifier (SID) (property: **Member (SID)**) of the



member. If the new member is a domain user account which is created in the same script, and multiple domain controllers exist, it is strongly recommended to use the security identifier to specify the new member. By default, the *Script Action: Create User (AD)* on page 3 generates a variable (%UserSid%) that holds the security identifier for the new user account. This variable can be used to specify the property: **Member (SID) = %UserSid%**.

The reason behind this mechanism is the fact that internally, the network operating system will try to resolve a specified account name to find the security identifier when the account is added to the local group. This operation might fail in case different domain controllers are used to create the account and to find the security identifier.

#### Properties

##### 4.1.2.

Property name	Description	Typical setting	Remarks
Computer	The name of the computer that contains the local group. The computer can be a workstation, domain member server, domain controller or workgroup member. The name must be specified as a NETBIOS or DNS name. If this property is specified, the property 'Domain' is ignored.		When specified, the <b>Domain</b> property is ignored.
Domain	The name of the domain that contains the local group. The domain must be specified as a NETBIOS or DNS name. If the group is not a domain local group, this property must not be specified.		Only used if the <b>Computer</b> property is not specified.

Local group name	The name of local group. The name must be specified as a single text field, for instance 'Administrators'. Preceding domain and computer names and (back)slashes are removed.		Mandatory property. Name of the local group to which the new member is added.
Member (SID)	The new group member, specified as a (variable holding a) security identifier (SID). When the SID of the new member is available, it is recommended to use this property to specify the new member. If this property is specified, the property 'Member (name)' is ignored.		When specified, the <b>Member (name)</b> property is not used. See <b>Deployment</b> section for more information.
Member (name)	The new group member specified by the name of the new member. When the SID of the new member is available, it is recommended to use property 'Member (SID)' instead. When the SID is not available, you should use this property. The group member can be a user account or global group. The name must be specified using syntax 'DOMAIN\\MEMBER' or 'MEMBER'.		Only used when the <b>Member (SID)</b> property is not used. See <b>Deployment</b> section for more information.
Error if already member	When set, no error is generated when the account is already a member of the local group. Default value: 'No'.	No	

**See also:**

*UMRA Basics* on page 3

*Script Action: Create User (AD)* on page 3

*Script Action: Set User Group Memberships (AD)* on page 56

*Script Action: Create local group*

### Function

Using this action you can create a local group on a server or workstation.

### Properties

Property Name	Description	Typical setting	Remarks
Computer Name	The name of the workstation or member server on which to create the local group		Used when creating a local group on a member server or workstation  May fail when setting local groups on a Domain controller. In that case specify the domain name rather than the computer name
Domain	The domain in which to create the group.	Not used	Typically used to create a domain local group on a domain controller.
Local Group name	The local group name	%GroupName%	This name is required,  The cannot be identical to any other user or group name on the domain or workstation being administered. It can contain up to 20 uppercase or lowercase characters, except for the following: " / \ [ ] : ;   = , + * < > . A group name cannot consist solely of periods (.) or spaces.

Comment	A text string, that will be shown in the Description field of the group in windows. The string can have any length.		
Error if group already exists	When set to 'Yes' an error will be generated if the group already exists	Yes	

**Remarks**

This action is mainly intended to create local groups on Workstations and member servers, outside of active directory.

To create local and or global groups in AD use *Script Action: Create group (AD)* on page 138 instead.

*Script Action: Remove group member*

**Function**

Removes the group member from a specific group.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts. With this action you can remove a group member from a specific group.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>
Group domain name	The domain name of the group from which the member must be removed. To identify the group, specify a value for either the preproperty "Group name" or "Group computer name".	NA
Group computer name	The computer name of the group from which the member must be removed (e.g. SERVER_A). The name of the computer can be a local computer, server or domain controller. To identify the group, specify a value for either the property "Group name" or "Group domain name".	NA
Group name	The name of the group as specified by the SAM account name (e.g. "Students"). To identify the group, specify a value for either the property "Group domain name" or "Group computer name".	NA
Member name	The name of the group member that must be removed. For global groups, please specify the SAM account name of the member. For local groups, you need to include the domain name (e.g. STUDENTS\Group_A)	NA
Global group flag	A flag which indicates if the group is a global group (= "Yes"), a domain group (= "No"), or a computer local group (= "No"). The default value is "Yes".	Yes

*Script Action: Set primary group (non AD)*

### Function

Sets the primary group. Can be used both in Windows NT and in Active Directory.

### Properties

Property name	Description	Typical setting	Remarks
Domain controller	The name of the domain controller that maintains the account. To determine the domain controller used to set the primary group, either this property or the property Domain must be specified.		
Domain	The name of the domain that maintains the account. To determine the domain controller used to set the primary group, either this property or the property Domain controller must be specified.		
Account name	The SAM account name of the account for which the primary group must be set.	%UserName%	
Group name	The name of the primary group		

### See also:

*Script Action: Set primary group (AD) on page 155*

## General user Actions

*Script Action: Edit user logon*

### Function

Edits the logon settings of an existing user account . The account is identified by a variable containing the User Object. Use the *Script Action: Get user (AD)* on page 31 to find the user first. For the user account, all regular attributes can be changed and/or reset.

### Deployment

This action is typically used as one of the main action to manage existing user accounts in Active Directory. You can use this action for a single change, for instance resetting the password of an account or multiple changes like home directory, profile directory and Active Directory attributes. To change the common name (full name) of a user account, you cannot use this action. Use the *Script Action: Move - rename user (AD)* on page 63 instead to do this.

For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the action *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

The **Edit user logon** action contains a large number of properties. As described above, the **User Object** property is used to identify the user account. Other properties are initially not specified. This means that the corresponding Active Directory attributes of the user account are not changed when the action is executed. Only when a property is specified, the attribute is updated in Active Directory.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
User Object	An data structure representing the user account. Use the action <i>Get user (AD)</i> on page 31 to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	See Deployment section.
Username	The SAM account name of the user for which you want to edit the logon settings.		You should only use this option when you are not using the %UserObject% variable. Instead of the %userObject variable an user account can also be identified by the <b>user name</b> and the <b>domain</b> name or the <b>domain controller</b> .
Domain	The domain in which the user account, for which you want to edit the logon settings, is located.		You should only use this option when you want to identify the user account by <b>username</b> and <b>domain</b> name.
Domain controller	The domain controller of the domain in which the user account, for which you want to edit the logon settings, is located.		You should only use this option when you want to identify the user account by <b>username</b> and <b>domain controller</b> .



Password generator	The specification how to generate passwords for the user account		<p>Specifies the method used to generate a password for the user account. These methods vary from simple (easy to remember) passwords to strong passwords. There are several predefined settings available.</p> <p>The resulting password will be stored in a variable. By default it is stored in the variable %Password%. This variable must be specified as the value for the <b>Password</b> property.</p>
Password	The password of the user account.		<p>Typically the name contained in the variable %Password% is generated by the <b>Password generator</b>. To create the same password for all users you can specify the password here directly. For example "test1234". You can also read the password from the input file.</p>
User must change password at next logon	The password is expired. Use this property to force the user to change the password at the next logon. Note that the user can logon using the current password.		<p>When set to <b>Yes</b> the <b>User cannot change password property</b> must be set to <b>No</b>.</p>

User cannot change password	The user cannot change password. When the user cannot change the password, only the administrator can change the password.		Valid specifications are <b>Yes</b> and <b>No</b> . This setting has no effect on members of the administrators group. When set to <b>Yes</b> , the <b>User must change password at next logon</b> property must be set to <b>No</b> .
Password never expires	The password should never expire on the account.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . This setting overrides the <b>Maximum Password Age</b> setting in the password policy for the domain/computer.
Account disabled	The user's account is disabled. If an user account is disabled, the account does exist but cannot be used to logon to the network.		
Unlock the account	Unlock an user account. When an account is locked it is temporarily impossible to log on to the network. An account gets locked when an incorrect password is specified.		Valid specifications are <b>Yes</b> and <b>No</b> . The default value is <b>No</b> . When set to <b>Yes</b> an locked account will be unlocked. This property can only be used when an account is locked.

**See also:**[Help on help](#)

*Script Action: Move - rename user (AD)* on page 63

*UMRA Basics* on page 3

*Script Action: Get user info*

Security administrators and managers frequently request a user account status report—that is, a report showing which accounts in a domain are active, which are locked out, and which are disabled. Active Directory (AD) user accounts have a bitmask attribute called `userAccountControl` that you can check to determine the user account status. Some flags of this attribute can easily be retrieved using an LDAP call in a generic table, but some other flags like "Account disabled" and "User must change password at next logon" cannot be retrieved this way. For these user flags you can use the **Get user info** script action.

Please note that in test mode, this function will not return any values.

#### Properties

Property Name	Description	Typical setting	Remarks
Domain		%Domain%	Often the domain name is used in many different actions, and is determined and stored in a variable previous to the action (e.g. %Domain%). The name of the domain can be either in DNS or NETBIOS style. (e.g. Tools4ever.com or TOOLS4EVER). For more information on how to specify the domain/OU/container in which the user account is created, see the <b>Remarks</b> section below.

Computer	The name of the computer that maintains the user account. This computer can be specified with a DNS or NETBIOS name. The computer can be a domain controller of a Windows NT4/2000/2003 domain, a member server of a domain or a workstation. If this property is specified, the 'Domain' property is ignored.		To specify the user account, specify properties Username and Domain or Computer.
Username	The name of the user account. The name equals the SAM account name of the user account.		To specify the user account, specify properties Username and Domain or Computer.
Full name	The full name of the user account.		
Description	A description associated with the user account		
Account disabled	Output only property. When this property is set to Yes, the account does exist but cannot be used to logon to the network		Based on property flag ACCOUNTDISABLE (hex value 0x0002).

No password required	Output only property. When this property is set to Yes, no password is required for the user account		Based on property flag PASSWD_NOTREQD (hex value 0x0020)
User cannot change password	Output only property. When this property is set to Yes, the user cannot change the password. Only the administrator can change the password.		Based on property flag PASSWD_CANT_CHANGE (hex value 0x0040)
Locked out	Output only property. When this property is set to Yes, the user account is currently locked out.		Based on property flag LOCKOUT (hex value 0x0010)
Password never expires	Output only property. When this property is set to Yes, the password for the account will never expire.		Based on property flag DONT_EXPIRE_PASSWD .

Relative identifier	Output only property. The relative identifier (RID) uniquely defines the user account within the domain		In Windows 2000, the relative identifier (RID) is the part of a security ID (SID) that uniquely identifies an account or group within a domain. Each newly created object in Active Directory is automatically assigned to an RID. Each domain controller has a pool of RIDs. If necessary, Windows adds to these pools in batches of 500. You can check the range of RIDs in a current pool using the system command <code>dcdiag /v /test:ridmanager</code>
---------------------	---	--	---

**See also:**

*Script Action: Get user (AD) on page 31*

*Script action: Terminal Services user settings*

**Function**

Sets the Terminal Services settings for a new or existing user account. The account either exists in a Active Directory or NT4 domain.

**Deployment**

This action is typically used in a script that is intended to:

- create new users in Active Directory or NT4 domains and to setup the Terminal Services settings for each individual account or
- to setup the Terminal Services for a number of existing user accounts.

For new user accounts, the action that creates the user account should precede this action. For new user accounts in Active Directory, it is strongly recommended to create the user account using server binding,

e.g. specify the domain controller both in this action and the action that creates the user account in Active Directory.

**Properties**

Property Name	Description	Typical setting	Remarks
User account	The name of user account for which the Terminal Services settings must be applied. The user account must be specified using the first part of the user logon name (j.smith@tools4ever.com -> j.smith) in Active Directory or the SAM account name (username) in Windows NT4 networks.	%Username%	

Domain Controller	The name of the domain controller that maintains the user account (DNS or NETBIOS style, e.g. server_1.tools4ever.com or SERVER_1). In case the user account is just created and multiple domain controllers exist, this property should equal the domain controller used to create the account. If this value is specified, the 'Domain' property is ignored.	%DomainController %	If the <b>Domain Controller</b> property is specified and the user account is created in Active Directory in the same script, you must specify the same domain controller in the action that creates the user account in Active Directory.
-------------------	--	---------------------	--



Domain	<p>The name of the domain (DNS or NETBIOS style, e.g. tools4ever.com or TOOLS4EVER) of the user account. If this property is specified and the 'Domain Controller' property is not specified, User Management Resource Administrator searches for an arbitrary domain controller of the domain. In case the user account is just created and multiple domain controller exist, this domain controller might not recognize the user as an existing user account. In this case it is advised to specify the property 'Domain Controller' instead. This property is ignored if a value is specified for the property 'Domain Controller'.</p>		
Profile path	<p>The Terminal Services Profile path. The profile is a roaming or mandatory user profile for use when the user logs on to a Terminal server. To enable a roaming or mandatory profile, type the network path in this form: \\server name\profiles folder name\user name. To assign a mandatory user profile, type the network path in this form: \\server name\profiles folder name\user profile name. The Terminal Services profile path is used for logging on to Terminal servers only. If you specify a profile path for logging on to Windows 2000, the path is also used for logging on to Terminal servers unless you specify a Terminal Services profile path here.</p>		

Home directory	The Terminal Services home directory. Each user on a Terminal server should have a unique home directory. This ensures that application information is stored separately for each user in the multiuser environment. You can specify a directory on the local server (example: C:\\Users\\%Username% - > C:\\Users\\johnw) or shared network directory (\\\\Server_A\\Users\\%Username% - > \\\\Server_A\\Users\\johnw). In the latter case, you also need to specify a value for the 'Home directory drive' property.		
Home directory drive	The Terminal Services home directory drive. Specify the drive letter (example: J:) mapped to the shared network directory specified for property 'Home directory'. In case you specify a local home directory, you should not to specify this property.		
Allow logon to terminal server	Specifies whether the user is permitted to log on to the Terminal server.	Yes	
End disconnected session (seconds)	Sets the maximum time that a disconnected session remains active on the server. If you specify this property, a disconnected session is reset after the time in seconds elapses. The value is specified in seconds. Do not specify this property if you don't want to reset a disconnected session on the server.		

Active session limit (seconds)	Sets the maximum duration for sessions in seconds. If you specify a duration, the session is disconnected or reset after the time elapses. Do not specify this property (or specify a value of 0 (zero)) to allow the connection to continue for an unlimited period.		
Idle session limit (seconds)	Sets the maximum idle time in seconds allowed before the session is disconnected or reset. If you specify a duration, the session is disconnected or reset after there has been no client activity for that period of time. Do not specify this property (or specify a value of 0 (zero)) to allow clients to remain idle indefinitely.		
Disconnect on connection broken - time-out	Disconnect the client when the connection to the server is broken for any reason, including a request, a connection error, or a session limit is reached. The client can reconnect to the session if needed. If you specify no, the session is reset. A reset session cannot be reconnected.	Yes	
Allow reconnection from any client	Specifies that Terminal Services allows reconnection to a disconnected session from any computer. This is the default setting. If you select 'No' a reconnection to a disconnected session is restricted to the computer that started the session. This option is supported only for Citrix ICA-based clients that provide a serial number when connecting."),	Yes	

User can specify initial program	Specifies whether the user can start any program. If you specify 'No' the program specified at property 'Logon program' runs automatically when the user logs onto a remote computer. Terminal server logs the user off when the user exits that program.	Yes	
Logon program	The path and file name of the application that you want to start when the user logs on to the Terminal server.		
Logon program working directory	The working directory path for the application that you want to start when the user logs on to the Terminal server.		
Connect client drives at logon	This option is for ICA clients only. Specifies whether to automatically reconnect to mapped client drives.	Yes	
Connect client printers at logon	Specifies whether to automatically reconnect to mapped client printers.	Yes	
Default to main client printer	Specifies whether to automatically print to the client's default printer.	Yes	
Remote control	Specify the level to control or observe a user's session. If you do not specify value for this property, the remote control function is disabled.		
Callback enabled	Set this property to 'Yes' if you want to enable the Terminal Server callback function. By default, (or when you specify 'No'), this function is disabled.		

Fixed callback phone number	Set this property to 'Yes' if you want the Terminal Server to callback at a default fixed phone number. You need to specify the number for property 'Callback phone number'.		
Callback phone number	Specify the callback phone number. If you set this, value, you should also set the value of properties 'Callback enabled' and 'Fixed callback phone number' to 'Yes'.		

*Script Action: Get terminal services user settings*

### Function

Retrieves the Terminal Services settings for an existing user account. The account either exists in a Active Directory or NT4 domain.

### Properties

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing the user account. The property is used to identify the user account and is normally generated as a variable by a previous script action.	%UserObject%	Use only with Windows 2003 or higher!
User account	The user account specified by the domain SAM account name	%DomainUserName	This property must be used with Windows 2000 and Windows NT4 networks.

Profile path	Roaming or mandatory profile path to use when the user logs on to the Terminal Server.		
Home directory	Home directory for the user. Each user on a Terminal Server has a unique home directory. This ensures that application information is stored separately for each user in a multi- user environment.		
Home directory drive	Home drive for the user. In a network environment, this property is a string containing a drive specification to which the home directory is mapped.		
Allow logon to terminal server	Specifies whether the user is permitted to log on to the Terminal server.	Yes	
Enable remote control	Value that specifies whether to allow remote observation or remote control of the user's Terminal Services session.		

Max. disconnection time	Maximum amount of time, in minutes, that a disconnected Terminal Services session remains active on the Terminal Server. After the specified number of minutes have elapsed, the session is terminated.		
Max. connection time	Maximum duration of the Terminal Services session, in minutes. After the specified number of minutes has elapsed, the session can be disconnected or terminated.		
Max. idle time	Maximum amount of time, in minutes, that the Terminal Services session can remain idle. After the specified number of minutes has elapsed, the session can be disconnected or terminated.		
Reconnection action	Value that specifies whether to allow reconnection to a disconnected Terminal Services session from any client computer.		

Broken connection action	Value that specifies the action to take when a Terminal Services session limit is reached.		
Client drives at logon	Value that specifies whether to reconnect to mapped client drives at logon.		
Client printers at logon	Value that specifies whether to reconnect to mapped client printers at logon.		
Default printer	Value that specifies whether to print automatically to the client's default printer.		
Working directory	Working directory path for the user.		
Initial program	Path and file name of the application that the user wants to start automatically when the user logs on to the Terminal Server.		

*Script Action: Dial-in user settings*

### **Function**

Sets the dial-in setting for an active directory user account. This function is used for remote access permissions to be explicitly allowed, denied, or determined through remote access policies.

### **Deployment**

This action is typically used in a script that is intended to create new user accounts or manage existing user accounts. The user account for which



the dial-in setting should be set is identified by the properties **User account** and **Domain Controller**. To execute this action successfully, these two properties must have a valid value. Different settings can be applied to increase the security. Dial-in options should always be set as securely as possible.

#### Properties

Property Name	Description	Typical setting	Remarks
User account	The name of user account for which the Dial-in settings must be applied. The user account must be specified using the first part of the user logon name (j.smith@tools4ever.com -> j.smith) in Active Directory or the SAM account name (username) in Windows NT4 networks.	%UserName%	
Domain Controller	The name of the domain controller that maintains the user account (DNS or NETBIOS style, e.g. server_1.tools4ever.com or SERVER_1). In case the user account is just created and multiple domain controllers exist, this property should equal the domain controller used to create the account.	%DomainController%	

Allow access	Specifies whether dial-up, virtual private network (VPN), authentication switch, or wireless access is allowed for the user.	Yes	This option should be cleared when you want to use the 'Use Remote Access Policy' option.
Use Remote Access Policy	Specifies whether a remote access policy is used for setting dial-up, virtual private network (VPN), authentication switch, or wireless access properties for the user.		When set to 'Yes' , the 'Allow access' option should not be set.
No Callback	If this property is enabled (default), the RAS server doesn't call the caller back during the connection process.	Yes	Only one of the three callback options (No Callback, Callback - Set by Caller, Callback - Always Callback preset phone number) should be set to Yes.
Callback - Set by Caller	Specifies whether a user can set the callback number.	No	Only one of the three callback options (No Callback, Callback - Set by Caller, Callback - Always Callback preset phone number) should be set to Yes.

Callback - Always Callback preset phone number	Specifies whether a preset phone number is used for the callback function.	No	Only one of the three callback options (No Callback, Callback - Set by Caller, Callback - Always Callback preset phone number) should be set to Yes. When set to 'Yes' a Callback phone number should be set.
Callback phone number	Specifies the number the server should call back to.		This option should only be used when the 'Callback - Always Callback preset phone number' option is set to Yes.

**See also:**

*UMRA Basics* on page 3

**4.1.3. Active Directory****Script Action: Create object (AD)****Function**

Creates an AD object

**Deployment**

This action is typically used for creating non-user objects in the AD (e.g. an OU)

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Remarks</b>
Domain	The name of the domain where the object will be created	If you specify a value for this property, please do not specify a value for property <b>LDAP container</b> since this specification takes precedence.
Organizational Unit-Container	The name of the Organizational Unit-container where the object must be created.	If you specify a value for this property, you should also specify a value for the <b>Domain</b> property. In that case, do not specify a value for the property LDAP container since this specification takes precedence.
LDAP container	The LDAP name of the OU or container where the object must be created.	You must specify a value either for this property or values for the properties <b>Domain</b> and <b>Organizational Unit-Container</b> . If values for both methods are specified, this method takes precedence.
Domain (controller)	The name of the domain controller or domain, used to access the domain, container or OU where the object must be created.	Optional. If this value is not specified or if the name of a domain is specified, the application creates the account on a domain controller that is to be determined by Active Directory (serverless binding). If a domain controller is specified, the account is explicitly created on the specified controller (server binding). In both cases ActiveDirectory will replicate the account information to all domain controllers in the ActiveDirectory forests and domains.
Class name	The object type to be created. Specify as the LDAP class name.	

Common Name	The CommonName corresponds with the Common Name of the object. This name defines the contact in an OU and must be unique. You can use the Name generation algorithm to make the name automatically unique.	
Object Distinguished Name	Output only. The Object Distinguished Name of the just created object	
Active Directory Object	An internal data structure representing the object. This property is an "output only" property and is generated automatically. This property can be used in other script actions.	Note there is a difference between the "Active Directory Object", and the similar "User object" generated by the "create user (AD) action".  Some Script actions allow you to specify either the "User Object", or the "Active Directory Object" as an input value. Make sure that you fill the correct corresponding input parameter

**Script Action: Delete Object (AD)****Function**

Deletes an existing AD object

**Deployment**

This action is typically used for deleting existing non-user objects in the AD (e.g. an OU)

**Properties**

The object to delete must be specified by one of the following properties:

Property Name	Description
Active Directory Object	An internal data structure representing the object. This property is an "output only" property and is generated automatically. This property cannot be used in other script actions.
LDAP name	The LDAP name of the object to be deleted.
SAM account name	The SAM account name (Pre- Windows 2000) of the object you want to delete.

**See also:**

*Script Action: Create object (AD)* on page 117

**Script Action: Get attribute (AD)****Function**

Gets the value of an attribute of an Active Directory user account or other object. The attribute is specified by the LDAP display name of the attribute. For the most common properties, the LDAP name can be selected from a list.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts or other Active Directory objects. Once the attribute is found for the object, the attribute value is saved in a variable that can be used by subsequent actions of the script. The actions supports multi-value attributes: When an attribute has multiple values, the values can be stored as multi-values or converted to a single value.

The attribute can be obtained from any Active Directory object. In most scripts, the Active Directory object is an user account. The Active Directory object must be specified as a variable. This variable is used for property User Object or property Active Directory Object. The *Script Action: Get user (AD)* on page 31 can be used to set the value for the variable used for the **User Object** property. For the **Active Directory Object** property, the action *Script Action: Get Object (AD)* on page 145 can be used. Either one of the two properties **User Object** and **Active Directory Object** must be used.

#### Properties

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing a user account. If you want to obtain the property of a user account object, you can use this property to specify the Active Directory object for this action. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action, for example <i>Script Action: Get user (AD)</i> on page 31.
Active Directory Object	A data structure representing a Active Directory object for which an attribute must obtained. This property can only be used as a input variable. Earlier in the script, another script action must have generated the value for this variable.		

Convert to text	<p>If set to Yes, the value is converted to text.</p> <p>If not, the value is converted to one of the UMRA supported data types.</p>	Yes	See section <b>How attribute values are stored</b> below
Multi-value flag	<p>Specifies how to convert an AD multi-value attribute to the resulting UMRA variable.</p> <p>If the AD value is multi-valued, and this flag is Yes, the value is stored as an list or an table type variable.</p> <p>If the AD value is multi-valued. and this flag is No, the values are converted and stored in single text variable.</p>	No	See section <b>How attribute values are stored</b> below



LDAP attribute display name	The LDAP name of the attribute. The name identifies the attribute of the Active Directory object. For a number of well-known attributes, the LDAP name can be selected from a list but you can specify any other valid name.		A LDAP attribute has several names. In the Windows 2003/2000 schema, for instance the common name and the LDAP-Display-Name are used. (example: for the NT-style name of a user, the common name is 'SAM-Account-Name' and the LDAP display name is sAmAccountName. Note that these names are case sensitive.
Error if no attribute found	Generate an error for this script action if the specified attribute is not found.	Yes	
Error if empty	Generate an error for this script action if the attribute is found but attribute value is empty.	Yes	
Attribute value	The value found for the attribute. This property is an 'output only' property and is generated by the application automatically. By default, the value for this property is stored in variable %AttributeValue%.		In most cases, you must specify a output variable for this property. Otherwise, the value of the attribute cannot be used in other script actions.

#### How attribute values are stored

Active Directory contains many different data types. In UMRA, the following data types are supported:

- text
- numeric
- date-time
- long integer
- Boolean

The way in which the values of output variables are stored, depends on your settings. The table below provides an overview of the various possible settings and the resulting effect for the way in which the output variable is stored.

An instance of a single-valued attribute can contain a single value (e.g. givenName, surname, title). An instance of a multi valued attribute (e.g. group membership lists) can contain either a single value or multiple values. Depending on the Multi-value flag and **Convert to text** properties (Yes or No), the data types will be stored as follows:

If Convert to text is	And Multivalue is	Then the variable is stored as type
Yes	No	text
Yes	Yes	text list
No	Yes	table
No	No	single unconverted data

If you are not sure what the original data type of an attribute value is, the best option is to choose the table type (original value is not converted).

**See also:**

*UMRA Basics* on page 3

*Script Action: Set attribute (AD)* on page 124

**Script Action: Set attribute (AD)**

**Function**

Dit is de versie in master helpSets the value of an attribute of an Active Directory object. The attribute to set is specified by the LDAP display

name of the attribute. For the most common properties, the LDAP name can be selected from a list. There are several options to specify which changes are made. You can for example skip or overwrite an attribute when the attribute value is already present.

### Deployment

This action is typically used in a script that is intended to manage existing objects and update a particular Active Directory attribute. You can manage an Active Directory user account by the %UserObject% variable (Use *Script Action: Get user (AD)* on page 31 to obtain the variable) or every other Active Directory object by the %ActiveDirectoryObject% variable (Use *Script Action: Get object (AD)* on page 145 to obtain the variable).

### Properties

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing a user account. If you want to set the property of a user account object, you can use this property to specify the Active Directory object for this action. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action, for example <i>Script Action: Get user (AD)</i> on page 31.

Active Directory Object	A data structure representing an Active Directory Object. If you want to set the property of an Active Directory Object, you can use this property to specify the Active Directory object for this action. Use the action 'Get object (AD)' to find the object in Active Directory and setup the variable that contains the 'Object'.	%ActiveDirectoryObject%	The Active Directory Object must always be specified as a variable. This variable must have been set by a previous script action, for example <i>Script Action: Get object (AD)</i> on page 145.
Active Directory object LDAP name	The full LDAP name of the target Active Directory object. This object can be any object in Active Directory.		Example: cn=John Williams, ou=Schools, dc=Tools4ever, dc=Com

LDAP attribute display name	The LDAP name of the attribute. The name identifies the attribute of the Active Directory object. For a number of well-known attributes, the LDAP name can be selected from a list but you can specify any other valid name.		A LDAP attribute has several names. In the Windows 2003/2000 schema, for instance the common name and the LDAP-Display-Name are used. (example: for the NT-style name of a user, the common name is 'SAM-Account-Name' and the LDAP display name is sAmAccountName. Note that these names are case sensitive.
-----------------------------	--	--	---

Attribute value	The value of the attribute. The value must be specified as a text value. When the attribute value is multi-value, the multi-value flag should be set to 'Yes'		<p>To include carriage return line feed characters, specify the escape character sequences according to the following values. At runtime, the values will be replaced into the real values:</p> <p>[\\r]: Carriage return</p> <p>[\\n]: Line feed</p> <p>[\\r\\n]: Carriage return - line feed</p> <p>[\\t]: Tab</p> <p>By default, [\\r\\n] is used to move to the beginning of the next line.</p> <p>Example to specify an address:</p> <p>New York[\\r\\n]USA</p> <p>will result in:</p> <p>New York USA</p>
Skip if new value empty	Default value: 'No'. Specify 'Yes' to ignore this action if the new attribute value is empty. In this case, the attribute is not changed. If this property is not specified or set to 'No', the target attribute is always updated.),		The new attribute value is empty when the text value contains no characters. If the value contains a single blank character, it is considered not empty.

Multi-value flag	Default value: 'No'. This value must be set to 'Yes' when multi-value attributes should be set.		
Append versus update multi-value flag	Default value: ' No'. When set to 'Yes' the current values will stay the same. When set to 'No' the current values will be replaced with the specified values		

**See also:**

*Script Action: Get user table (AD) on page 51*

**Script Action: Delete attribute value (AD)****Function**

This script action deletes the value of an attribute for a specified user object. Note that you cannot delete the value of attributes with a "System Only" flag in the attribute definition (e.g. memberOf).

**Deployment**

This action can be used to clean up attribute values which are no longer used. This could be the case for instance, if you have a company where a pager is no longer used. The attribute values for the pager numbers can then be deleted.

### Properties

Property Name	Description	Typical setting
Active Directory object	A data structure representing the Active Directory object. This object can be obtained using either the <i>Script Action: Get user (AD)</i> on page 31 or <i>Script Action: Get Object (AD)</i> on page 145..  Use either this property or the property <b>Active Directory path</b> to specify the Active Directory object.	%UserObject%
Active Directory path	The full LDAP name of the Active Directory object of which the attribute must be deleted. Use either this property or Active Directory object to specify the Active Directory object.	
LDAP attribute display name	The LDAP display name of the attribute (e.g. telephoneNumber)	

### See also:

*Script Action: Get user (AD)* on page 31

*Script Action: Get Object (AD)* on page 145

### Script Action: Remove SID history

#### Function

This script action allows you to remove the SID history of the object contained in the variable %ActiveDirectoryObject%.

#### Deployment

This script action can be used as part of a complete cleanup operation to remove user accounts.



**Property**

Property name	Description	Typical setting
Active Directory Object	A data structure representing a Active Directory object for which the SID history must be removed. This property can only be used as a input variable. Earlier in the script, another script action must have generated the value for this variable.	The data structure contained in %ActiveDirectoryObject% is obtained as a result of either the action <i>Get user (AD)</i> on page 31 or <i>Get object (AD)</i> on page 145.

**Script Action: Update group memberships (AD)****Function**

Adds, removes or synchronizes an Active Directory account (user, contact, group etc.) with a number of Active Directory groups. An update of multiple group membership can take place. Multiple groups can be specified for the Active Directory account. Both the Active Directory account and groups must exist.

**Deployment**

The action can execute one of three main tasks. For each of these tasks, multiple groups can be specified. The account can be any Active Directory object that can become a member of Active Directory groups, including user accounts, groups account etc. The three main tasks are:

1. **Add an account to a number of specified Active Directory groups.**  
The account can already be a member of the specified groups or other groups. The account is only added to the specified groups if the account is not already a member of the group. The account is not removed from any group;
2. **Remove an account from a number of specified Active Directory groups.** For each specified group, the action checks if the account is a member of the group. If this is the case, the account is removed from the group. No other updates take place;

### 3. **Synchronize an account with a number of Active Directory groups.**

On completion, the account only is a member of the specified groups. To accomplish the synchronization, group memberships can be removed and/or added.

In Active Directory, a user account always is a member of a **primary group**. Also, an account cannot be removed from its primary group unless another group is assigned as the primary group. By default, the primary group is Domain Users. With this action, the primary group is ignored. When using this action, do not remove an account from its primary group and when synchronizing, do not include the primary group in the synchronization list. Even when the the synchronization list does not contain the primary group, the action will not remove the account from the primary group. If the synchronization list does contain the primary group, an error is generated when the action is executed.

#### Properties

Property Name	Description	Typical setting	Remarks
Active Directory account object	An Active Directory object for which the group memberships are updated.	%ActiveDirectoryObject% %	The value of this variable should be obtained from an other action. This value can be obtained from script actions: Create user (AD), Create contact (AD), Get User (AD) or Get object (AD) etc. You should make sure the export variable of these actions is the same as the import variable of the property (default: %ActiveDirectoryObject% )

Add list	If the account must be added to a number of groups, specify this property. See the Remarks section for more information.		Note: specify only one of the properties 'Add list', 'Remove list' and 'Sync list'.
Remove list	If the account must be removed from a number of groups, specify this property. See the Remarks section for more information.		Note: specify only one of the properties 'Add list', 'Remove list' and 'Sync list'.
Sync list	If the account group memberships must be synchronized with a number of groups, specify this property. See the Remarks section for more information.		Note: specify only one of the properties 'Add list', 'Remove list' and 'Sync list'.

Binding information	Optionally: The binding information used to access the specified groups and Active Directory account. See the Remarks section for more information.		If this property is not specified, the action uses the binding information specified for each group, or LDAP:// if the group specification does not include binding information. If the property is specified, the binding information is used unless a group is specified with its own binding information.
---------------------	---	--	--

#### Remarks

Specify only one of the properties 'Add list', 'Remove list' and 'Sync list'. The task executed by the action depends on which property is specified: **Add**, **Remove** or **Sync**. Multiple groups can be specified in different ways as described below. The groups must be specified using the distinguished names, optionally including binding information. Valid specifications are:

`cn=grp_a,ou=org_a,dc=domain,dc=com`

`LDAP://cn=grp_a,ou=org_a,dc=domain,dc=com`

`LDAP://dc.domain.com/cn=grp_a,ou=org_a,dc=domain,dc=com`

Note that if a group is specified with binding information, the binding information overrides the value of optional property **Binding information**. So to use the value of property **Binding information** for all groups, specify each group with its distinguished name: `cn=Group...`

For each of the lists, the corresponding properties can be specified as follows:

1. Normal text, specifying a single group. Example:  
`cn=GroupA,dc=domain,dc=com;`
2. Normal text, specifying multiple groups, each group quoted with double quotes and individual group entries separated by comma's:

```
"cn=GroupA,dc=domain,dc=com","cn=GroupB,  
dc=domain,dc=com";
```

3. Normal text, specified as a text variable. The value of the text variable can be specified as described in options 1 and 2. Example: %GroupNames%;
4. Text list (specified as a variable): The variable contains a text list value, each list entry specifying a single group. Example: %GroupList%
5. Table (specified as a variable): The table should contain a least a single column, with the first column specifying a single group in each row. Example: %GroupTable%.

### **Script Action: Set group membership (AD)**

#### **Function**

Makes an Active Directory object a member of specified Active Directory universal, domain global or domain local groups. An update of the group membership will take place. The group membership will be added to the 'Member Of' list of the Active Directory object.

#### **Deployment**

This action is typically used in a script that is intended to manage existing objects in active directory. This action can be used to set group memberships for every object in your Active directory.

In this action the Active Directory Object is identified by a property value. You should either provide a data structure provided by an other action (Property: Active Directory Object) or provide the object distinguished (Property: Active Directory name). When you want to add the object to multiple groups, the Group names must be obtained from a multi-text variable *Script Action: Manage multi-text value variable* on page 558.

### Properties

Property Name	Description	Typical setting	Remarks
Active Directory Object	An Active Directory object for which the group memberships are updated.	%ActiveDirectoryObject%	The value of this variable should be obtained from an other action. This value can be obtained from script actions: Create user (AD), Create contact (AD), Get User (AD) or Get object (AD). You should make sure the export variable of these actions is the same as the import variable of the property (default: %ActiveDirectoryObject%)
Active Directory name	The object distinguished name of the Active Directory object.		You should use either the Active Directory Object (%ActiveDirectoryObject%) to identify the object or the Active Directory name. The active directory name should be specified by the object distinguished name. Example:cn=Group1,ou=OrgUnit,dc=tools4ever,dc=com
Group names (variable)	The names of the groups of which the object becomes a member.		The group names should be specified by there full LDAP name. You should use a multi-text variable to set this property using the <i>Script Action: Manage multi-text value variable</i> on page 558. The group memberships are updated not reset. The specified object will remain member of earlier specified groups.

### See also:

*Script Action: Get Object (AD)* on page 145

**Script Action: Remove specific group memberships (AD)****Function**

Removes a specific group membership of an Active Directory user account. Unlike the *Script Action: Remove user group memberships (AD)* on page 60 it does not remove ALL user groups, but only a specific one.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts. With this action you can delete the user account from a specific group of which the account is a member. More specifically, you would be using this function if a user moves from department A to B in which case you will need to remove specific group memberships and add new ones.

**Properties**

Property Name	Description	Typical setting	Remarks
Group name (LDAP)	The full LDAP name of the group from which the membership must be updated. To specify the group, enter a value for either the property "Group name (LDAP)", "Group name (SAM account name)" or "Group object".	NA	Unique within OU
Group name (SAM account name)	The group name specified using the SAM account name (e.g. DOMAIN_A\Group_C). To specify the group, enter a value for either the property "Group name (LDAP)", "Group name (SAM account name)" or "Group object".	NA	Unique within domain

Group object	A data structure representing the group. To specify the group, enter a value for either the property "Group name (LDAP)", "Group name (SAM account name)" or "Group object".	NA	This value can only be generated as a variable resulting from a previous script action.
Account name	The LDAP name of the account from which the group membership must be removed (e.g. LDAP://DC_B/CN=Student,DC=Domain,DC=com)	NA	
Account object	A data structure representing the account from which the group membership must be removed. To specify the group member, enter a value for either the property "Account Name" or "Account object".	NA	This value can only be generated as a variable resulting from a previous script action.

### Script Action: Create group (AD)

#### Function

Creates a group in Active Directory. Using this action you can create Local groups, Global groups or Universal groups. The groups can be Security groups or Distribution groups. The groups can be placed in any container you specify. A description can be added to easily identify the group.

#### Deployment

This action is typically used for creating multiple groups. When building your Active Directory from the ground up, one of the first thing you



should do is create the groups of which the other Active Directory object will be members. Groups can be used to easily allow or deny users access to parts of the network.

#### Properties

Property Name	Description	Typical setting	Remarks
Domain	The domain in which to create the group.	%Domain%	Often the domain name is used in many different actions, and is determined and stored in a variable previous to the action ( e.g. %Domain%). The name of the domain can be either in DNS or NETBIOS style. (e.g. Tools4ever.com or TOOLS4EVER). For more information on how to specify the domain/OU/container in which the group is created, see the <b>Remarks</b> section below.
Organizational Unit- Container	The name of the Active Directory Organizational unit or other container in which to create the group.	Users	Specify the path of the organizational unit (OU) or container relative to the domain. To specify OUs in OUs, use the full path relative to the domain, separated by slashes: OU/ChildOU/GrandChildOU. Examples: <b>students</b> or <b>students/group1</b> . For more information on how to specify the domain/OU/container in which the group is created, see the <b>Remarks</b> section below.

LDAP container	Optional: The LDAP name of the container in which to create the group.		<p>Optionally specifies name of the Active Directory container in which the group is created directly by means of its LDAP name (Example: CN=users, DC=tools4ever,DC=com Example: OU=Group1, OU=Students, DC=tools4ever, DC=com)</p> <p>This specification can be used instead of the Domain and Organizational Unit-Container properties of this action. If specified, the specified LDAP Container takes precedence, and the Domain And Organization Unit-Container properties are ignored. For more information on how to specify the domain/OU/container in which the group is created, see the <b>Remarks</b> section below.</p>
----------------	--	--	---

Domain (controller)	Optional: The name of the domain controller or domain used to access the domain.		<p>If this value is not specified, the application creates the account on a domain controller that is determined by Active Directory (serverless binding). If a domain controller is specified, the account is explicitly created on the specified controller (server binding). In both cases, Active Directory itself will replicate the account information to all domain controllers in the forest automatically as required.</p> <p>Depending on the actual User Management Resource Administrator Script used, it may be necessary to specify a domain controller here. If an subsequent script action does an Active Directory query to obtain information of the newly created group, this query may occur before Active Directory has replicated the new information to other Domain Controllers. As a consequence, the query may fail to find the newly created group. When both actions however specify the same domain controller, the newly created group can be found.</p> <p>Often a requery of Active Directory by subsequent actions for the newly created group can be prevented by using the Group Object that is created by this action in subsequent actions, instead of the name of the group.</p>
------------------------	--	--	---

CommonName	The CommonName is the name of the group. This name is most commonly used in user interfaces.	%GroupName%	In this action the CommonName and SAM-Account-Name will be the same by default. To change this, you should create an other variable for one of the settings.
SAM-Account-Name	The group name(Pre-Windows 2000) without the (NETBIOS) Domain name.	%GroupName%	This name is required, also in domains that use solely Active Directory domain controllers.  A SAM-Account-Name cannot be identical to any other user or group name on the domain being administered. It can contain up to 20 uppercase or lowercase characters, except for the following: " / \ [ ] ;   = , + * < > . A SAM-Account-Name cannot consist solely of periods (.) or spaces.
Description	A text string, that will be shown in the Description field of the group in windows. The string can have any length.		
Local group	When set to 'Yes' the created group will be a (domain) local group.	No	One of the three groups (local, global and universal), must be set to 'Yes'.

Global group	When set to 'Yes' the created group will be a global group.	No	One of the three groups (local, global and universal), must be set to 'Yes'.
Universal group	When set to 'Yes' the created group will be a universal group.	No	One of the three groups (local, global and universal), must be set to 'Yes'.
Security group	When set to 'Yes' the created group will be a security group. When set to 'No' a distribution group will be created.	No	
No error if group already exists	When set to 'Yes' no error will be generated.	No	Warning: when set to 'Yes' some errors are ignored and scripts may not be completed correctly.
Group Object Distinguished Name	The Object Distinguished name of the just create group.	%GroupODN%	output only. Can be used as input in other actions where a Object Distinguished name is required.
Group Object	An internal data structure representing the group. this property will only give an output. this output can be used in other script actions.		This script action has an output variable (default: %GroupObject%). This variable can be used in other script actions.

**Remarks****Domain / OU / Container / LDAP -specification**

User Management Resource Administrator supports several methods to specify the entity (domain, OU or container) in which the group will be created. These methods differ in the way the property values are specified. The properties involved are: Domain, Organizational Unit-Container, LDAP container. Depending on your network environment and input data, you should choose the method that fits best:

Properties specified	Properties not specified	Example	Description
Domain Organizational Unit-Container	LDAP container	Domain: TOOLS4EVER or tools4ever.com Organizational Unit-Container: STUDENTS/GROUP1	This is most easy method to create groups in OU's. To create the group, User Management Resource Administrator will automatically compose the LDAP name of the container to create the group.
Domain	LDAP container Organizational Unit-Container	TOOLS4EVER or tools4ever.com	Use this method only, to create groups in the domain root. No OU is involved.
LDAP container	Domain Organizational Unit-Container	OU=Group1, OU=Students, DC=tools4ever, DC=com	Use this method if you want to specify the OU directory using the LDAP format. If this property is specified, the Domain and Organizational Unit-Container properties are ignored.

**See also:**

*UMRA Basics* on page 3

**Script Action: Get Object (AD)****Function**

Accesses an object in Active Directory. The action is used always in combination with other subsequent actions. Once the object is found, an internal data structure representing the group is setup. This structure is stored in a variable (%ActiveDirectoryObject%) that can be used by other actions.

**Deployment**

This action is typically used in a script that is used to manage, edit or delete existing Active Directory objects. When this action is executed successfully, the subsequent actions in the script have access to the object using the variable %ActiveDirectoryObject%.

**Properties**

Property Name	Description	Remarks
LDAP name	The full LDAP name of the object. The LDAP name is used to identify the Active Directory Object.	Example: cn=John Williams, ou=Schools, dc=Tools4ever, dc=Com
Active Directory Object	An internal data structure representing the object. This property will only give an output. The output can be used in other script actions.	This script action has an output variable (default: %ActiveDirectoryObject%). This variable can be used in other script actions.
Relative name (output)	The relative name of the object	Example: "CN=GroupA"
Class name (output)	The name of the class of the object according to the AD schema	
Active Directory path (output)	This string uniquely identifies the object in a network environment.	the object can always be retrieved using this path.
Parent AD path (output)	The string identifies the container of the object.	
Schema AD path (output)	The AD schema path of the class of this object.	

Unique object identifier (output)	The GUID of the object.	
-----------------------------------	-------------------------	--

**See also:**

*UMRA Basics* on page 3

**Script Action: Search object (AD)****Function**

Searches the Active Directory for one or more objects. For each object found, the object distinguished name is returned. For the search, you need to specify the environment (LDAP, GC, domain, ou, etc.) and the LDAP search string.

**Deployment**

This action is typically used in a script that is intended to manage existing user. The accounts can be specified by an Active Directory attribute. This action is then used to find the Active Directory user object. Next, the output distinguished name of the user account can be used to compose to full LDAP name. The resulting name is then used in the *Script Action: Get user (AD)* on page 31 to bind to the user account.

The search is performed in an environment you can specify. There are three options:

- **Search in the entire Active Directory:** The application first determines the root domain name of the Active Directory environment and then binds to Active Directory. To select, specify **LDAP** for the property **Search environment**.
- **Search in the global catalogue of Active Directory:** The application first determines the root domain name of the Active Directory environment and then binds to Active Directory. To select, specify **GC** for the property **Search environment**.
- **Search in a specific domain, organizational unit or container of Active Directory:** With this option you can limit the scope of



the search operation. To select, specify the full LDAP name of the object you wish to search in for the property **Search environment**. Optionally, you can specify the name of domain controller (NETBIOS or DNS format) computer that the application must use to bind to Active Directory. Example:  
LDAP://domaincontroller/OU=students,DC=domain,DC=com.

If you are searching for specific objects in Active Directory, you need to specify a filter with criteria that only match for the objects searched for. The filter is specified as a text string according to RFC 2254. Example: to search for a object of class User, (e.g. a user account) with a specific content for the attribute description (1234) the filter looks like this:

(&(objectClass=user) (&(description=1234)))

If you don't know how to specify the filter, please contact Tools4ever support ([www.tools4ever.com](http://www.tools4ever.com) <http://www.tools4ever.com/support/user-management-resource-administrator/documentation/manuals-and-guides/>, [support@tools4ever.com](mailto:support@tools4ever.com) <mailto:support@tools4ever.com>).

#### Properties

Property Name	Description	Typical setting	Remarks
Search environment	The search is performed in one of three possible environment: LDAP, GC or any other object. To search the entire Active Directory environment accessible from the local computer, specify the word LDAP (1). To search in the Global Catalog, specify the word GC (2). To search in any other environment, specify the LDAP binding string to access the object (3). Example: To search in a specify domain: LDAP://domain or LDAP://host. To search in a specific OU: LDAP://domaincontroller/OU=students,DC=domain,DC=com.	LDAP	See <b>Deployment</b> section

LDAP search Filter	The LDAP search filter according to RFC 2254. Example, to find user accounts with a specific description field 1234: (&(objectClass=user) (&(description=1234)))		
Error if nothing found	Generate an error for this script action if no matching objects are found.	Yes	
Error if multiple found	Generate an error for this script action if multiple matching objects are found.	Yes	
Search in child objects	Search in the specified environment and child objects, for example child domains.	Yes	
Number of objects found	The number of matching objects found. This property is an 'output only' property and is generated by the application automatically. By default, the value for this property is stored in variable %SearchResultCount%.		The number of objects found can be stored in a variable. By default, the name of this variable is <b>%SearchResultCount%</b> .
Object distinguished names	The distinguished names of the matching objects. This property is an 'output only' property and is generated by the application automatically. By default, the value for this property is stored in variable %SearchResults%.		The object distinguished names are collected for each matching object. These names are stored in a single variable. By default the name of the variable is <b>%SearchResults%</b> .

**See also:**

*Script Action: Get user (AD)* on page 31

*UMRA Basics* on page 3

**Script Action: Move - rename (AD)****Function**

Moves a user account in Active Directory to another OU or container within the same domain. Alternatively, you can also use this action to rename a user account in an organizational unit - container of Active Directory.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts. With this action you can execute 2 operations:

1. **Move user account(s) to other organizational units:** The user account can be moved to another organizational unit in the same domain. When the account is moved, the common name of the user account is not changed by default. The common name is part of the full LDAP name of the user account that uniquely identifies the user account in the organizational unit or container. Hence, the common name must be unique in the organizational unit. If you execute this action and move an account to an OU and a user account with the same common name already exists in the OU, the action will fail. Alternatively, you can rename the account (property **NewName**).
2. **Rename a user account:** With this action you can change the common name of the user account. The common name is part of the full LDAP name of the user account that uniquely identifies the user account in the organizational unit or container. Hence, the common name must be unique in the organizational unit. If the new common name is not unique, the action will fail and an error is generated.

You can also combine the 2 possible operations and both move and rename the user account. When you want to move the user account, you need to specify the destination organizational unit or container of the user account. If you only want to rename the user account, the destination organizational unit or container is not changed for the user account. To specify the destination organizational unit or container you have 2 options:

1. Specify properties **Domain** and **Organizational Unit-Container**:  
When moving user accounts to another organizational unit, you must specify the new name of the OU. If the domain is not changed, you don't need to specify property Domain. If you want to use this option, you don't need to specify the property **OU-Container LDAP name**
2. Specify property **OU-Container LDAP name**: If you use this option, you need to specify the full LDAP name of the destination organizational unit - container. Examples: ou=Schools, dc=Tools4ever, dc=Com, LDAP://ou=Schools, dc=Tools4ever, dc=Com, LDAP://domaincontroller/ou=Schools, dc=Tools4ever, dc=Com. With this option, you don't need to specify the properties **Domain** and **Organizational Unit-Container**.

#### Properties

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing the user account. Use the action 'Get user (AD)' to find the user account in Active Directory and setup the variable that contains the 'User Object'.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action, for example <i>Script Action: Get user (AD)</i> on page 31.

Domain	The name of destination domain (DNS or NETBIOS style, e.g. tools4ever.com or TOOLS4EVER) of the user account. If the domain name is not specified, the application assumes that the account is not moved across domains. When no destination Organizational Unit-Container is specified, the user account is not moved but renamed only.		Specification of this property is required only if you want to move and optionally rename the user account across domains.
Organizational Unit-Container	The name of the destination Organizational Unit-Container of the user account (example: Students or Students/Group1). When this property is not specified, the user account is not moved but renamed only unless the property 'OU-Container LDAP name' is specified.		Specification of this property is required only if you want to move and optionally rename the user account to another organizational unit or container.

OU-Container LDAP name	The full LDAP name of the destination Organizational Unit-Container (example: ou=Schools, dc=Tools4ever, dc=Com). When specified, the properties 'Domain' and 'Organizational Unit-Container' are ignored. When no destination Organizational Unit-Container is specified, the user account is not moved but renamed only.		Specification of this property is required only if you want to move and optionally rename the user account to another organizational unit or container.
Domain controller	Optional: The name of the domain controller, used to access to the domain, container or organizational unit where the account is moved to or where the account exists in case of a rename operation. This property 'helps' User Management Resource Administrator to access Active Directory.		
New name	The new name of the user account. The name is the name that identifies the user account in Active Directory e.g. the 'Common- Name'. If this property is not specified, the account is not renamed. To rename other names of user accounts, use the action 'Edit user (AD)'.		You only need to specify this property if you want to rename the user account, e.g. change the common name.

**See also:**

*Script Action: Move cross-domain (AD) on page 67*

*UMRA Basics* on page 3

### Script Action: Move cross-domain (AD)

#### Function

Moves an existing user object (users and computer accounts) from an OU in one domain to an OU in another domain.

#### Deployment

This action is typically used in a script that is intended to manage existing user accounts. When moving a user account to another domain, several restrictions apply:

1. The source and destination domain must be in the same forest of domains.
2. The destination domain must be in native mode.

#### Properties

The script action **Move cross domain (AD)** has the following properties:

Property Name	Description	Remarks
Source object	The <b>Source object</b> property is the LDAP name of the object to be moved in the original location (before the move).	<b>Important:</b> In case the source domain has multiple domain controllers, the domain controller with the role of RID master must be used to access the source account. Access to the source account is controlled by specifying a binding string as part of the LDAP name:  LDAP://server_rid_master.mydomain.com on page 63/ CN=<AccountToMove>, OU=<SourceOU>, DC=<mydomain>.DC=com.

Target container	The Target container property is used to specify the full LDAP name of the destination of the object. The container can be an organizational unit, domain or general container (e.g. Users). The container must be specified using a server binding string in DNS format: goldfish.marketing.TheFirm.com. This type of specification enforces the move operation to use Kerberos authentication instead of NTLM.	
New name	New name only has to be specified if the (common) name of the user account changes. If not, it can be left unspecified.	

**See also:**

*Script Action: Move - rename user (AD)* on page 63

*UMRA Basics* on page 3

**Script Action: Get primary group****Function**

Gets the primary group

**Deployment**

The user's primary group applies only to users who log on to the network from a Macintosh client or those users running POSIX-compliant applications. Unless you are using these services, the default primary group is **Domain Users**.



**Properties**

Property Name	Description	Typical setting	Remarks
Active Directory object	A data structure representing the Active Directory object for which the primary group is updated.	NA	This value can only be specified as a variable resulting from a previous script action.
Primary group name	The name of the primary group (e.g. DOMAIN_A\Students).	NA	

**Script Action: Set primary group (AD)****Function**

Sets the primary group in an Active Directory environment.

**Deployment**

The user's primary group is only relevant for users who log on to the network from a Macintosh client or who are running POSIX-compliant applications. Unless you are using these services, there is no need to change the primary group from Domain Users, which is the default value.

**Properties**

Property Name	Description	Typical setting	Remarks
Active Directory object	A data structure representing the Active Directory object for which the primary group is updated.	NA	This value can only be specified as a variable resulting from a previous script action.
Primary group name	The name of the primary group (e.g. DOMAIN_A\Students).	NA	

#### 4.1.4. Exchange

##### Exchange 2000/2003

*Script Action: Create Exchange Mailbox (2003/2000)*

##### Function

Creates a Exchange mailbox for an Active Directory user account. This action supports MS Exchange versions 2003 and 2000.

##### Deployment

This action is typically used in a script that is intended to create new users in Active Directory, after creation of the actual user account with *Script Action: Create User (AD)* on page 3. It can also be used for modifying existing accounts.

##### Properties

Property Name	Description	Typical setting	Remarks
User Object	Internal application object representing the user account for which a mailbox must be created.	%UserObject%	The User Object must always be specified as a variable. This variable must have been set by a previous script action. For example the script action <b>Create user (AD)</b> will by default fill the variable %UserObject% with the User Object of the created user.
Exchange server	The name of the Exchange server on which the mailbox is created. It can be specified either in DNS-style or in NT4-style.	%ExchangeServer%	

Mailbox store	Optional: The LDAP name of the mailbox store.	<not specified>	A <b>Mailbox store</b> is required to create an Exchange mailbox. When this property is not specified, User Management Resource Administrator tries to determine the mailbox stores that exist on the specified <b>Exchange server</b> . When only one mailbox store is found, this mailbox store is used for the Exchange mailbox. By default, only one mailbox store is setup when MS Exchange is installed. If multiple mailbox stores exist on the Exchange server, you must explicitly specify this property.
Domain controller	Optional: The name of the <b>Domain controller</b> used to access the Active Directory.	<not specified>	Exchange information is stored in Active Directory. Depending on the logged on user account, and the network domain configuration, it may be necessary to specify this property. For instance, if you are logged in a trusted NT4 domain and are creating mailboxes in a Windows 2003/2000 environment, you must specify the name of a domain controller of the Windows 2003/2000 domain of the user account for this property. This property is used only to enable access to Active Directory.
Alias	Optional: The Alias property specifies the Alias used for E-mail address generation.	<not specified>	By default E-mail addresses are generated based on the name of the user account. The value is setup by MS Exchange automatically.

E-mail addresses	Optional: The explicit E-Mail addresses for the Exchange mail box.		<p>By default E-mail addresses are generated automatically when the mail box is created. By specifying this property you can overrule this setting and specify additional E-mail addresses.</p> <p>Overruling of automatically generated addresses only occurs for the E-mail types that are explicitly set. That is, if your Exchange server configuration default generates both SMTP and X400 addresses, and the this property specifies only SMTP addresses, the X400 addresses will still be generated as specified on the Exchange server itself.</p> <p>Specify the E-mail address using the format (E-mail-type):(E-mail-Address). To specify the primary address, the E-mail-type must be in capitals. There must be exactly one primary E- mail address of each E-Mail type when used.</p> <p>Example:</p> <p>SMTP:J.Smith@tools4ever.com smtp:John@tools4ever.com</p>
Auto-update E-mail addresses	When this is set to YES Exchange will automatically generate E-mail addresses according to the Exchange recipient policy for the account.	YES	

Hide from address book	When set to YES, the user's mailbox does not show in address books.	NO	
------------------------	---	----	--

**See also:**

*Script Action: Edit Exchange mailbox (2000/2003)* on page 159

*Script Action: Delete Exchange mailbox (2000/2003)* on page 168

*Script Action: Manage Exchange recipient mail addresses (2000/2003)* on page 169

*Script Action: Modify Exchange mailbox permissions* on page 162

*Script Action: Move Exchange mailbox* on page 167

**Help on help**

*UMRA Basics* on page 3

*Script Action: Edit Exchange mailbox (2000/2003)*

**Function**

Edits the existing Exchange 2003/2000 mailbox of an user account. The user account and mailbox must already exist. To edit additional attributes of the user account, use the *Script Action: Edit user (AD)* on page 37.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts and mailboxes. For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

**Properties**

Property Name	Description	Typical setting	Remarks
User Object	A data structure representing the user account. The property is used to identify the user account for the mailbox and is normally generated as a variable by a previous script action ('Creating user (AD)').	%UserObject%	
Alias	The Alias property specifies the Alias used for E-mail address generation.		By default E-mail addresses are generated based on the name of the user account. The value is setup by MS Exchange automatically.

E-mail addresses	The E-mail addresses specified for the Exchange mailbox. By default, the E-mail addresses are generated automatically when the mailbox is created. By specifying this property you can configure additional E-mail addresses.		<p>By default E-mail addresses are generated automatically when the mail box is created. By specifying this property you can overrule this setting and specify additional E-mail addresses.</p> <p>Overruling of automatically generated addresses only occurs for the E-mail types that are explicitly set. That is, if your Exchange server configuration default generates both SMTP and X400 addresses, and the this property specifies only SMTP addresses, the X400 addresses will still be generated as specified on the Exchange server itself.</p> <p>Specify the E-mail address using the format (E-mail-type):(E-mail-Address). To specify the primary address, the E-mail-type must be in capitals. There must be exactly one primary E-mail address of each E-Mail type when used.</p> <p>Example:</p> <p>SMTP:J.Smith@tools4ever.com smtp:John@tools4ever.com</p>
Auto-update E-mail addresses	The E-mail addresses for the Exchange mailbox can be generated according to the recipient's policy by specifying this option.		

Hide from address book	The property specifies whether the recipient is displayed in the address book.		
------------------------	--	--	--

## 4.2.

### See also:

*Script Action: Create Exchange Mailbox (2003/2000) on page 156*

*Script Action: Delete Exchange mailbox (2000/2003) on page 168*

*Script Action: Manage Exchange recipient mail addresses (2000/2003) on page 169*

*Script Action: Modify Exchange mailbox permissions on page 162*

*Script Action: Move Exchange mailbox on page 167*

Help on help

*UMRA Basics on page 3*

*Script Action: Modify Exchange mailbox permissions (2000/2003)*

### Function

Modifies the permissions of an existing Exchange 2003/2000 mailbox. The mailbox and user account must exist.



## Deployment

This action is typically used in a script that is intended to manage existing user accounts and mailboxes. With this action permissions of **the mailbox** can be added and removed. For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

With this action you can perform the following functions:

1. Add permissions for another account to the mailbox.
2. Delete permission for a specific account from a mailbox
3. Set specific mailbox permissions

## Properties

Property Name	Description	Typical setting	Remarks
User Object	An data structure representing the user account. The property is used to identify the user account for the mailbox and is normally generated as a variable by a previous script action ('Creating user (AD)').	%UserObject%	This property specifies the mailbox that must exist. The mailbox can be created with other actions. (see <i>Script Action: Create Exchange Mailbox (2003/2000)</i> on page 156) for more information.
Permission: Delete mailbox storage	Set this property to 'Yes' if you want to add the permission 'Delete mailbox storage'.		One of the standard permissions you can add to the mailbox.
Permission: Read permissions	Set this property to 'Yes' if you want to add the permission 'Read permissions'.		One of the standard permissions you can add to the mailbox.

Permission: Change permissions	Set this property to 'Yes' if you want to add the permission 'Change permissions'.		One of the standard permissions you can add to the mailbox.
Permission: Take ownership	Set this property to 'Yes' if you want to add the permission 'Take ownership'.		One of the standard permissions you can add to the mailbox.
Permission: Full mailbox access	Set this property to 'Yes' if you want to add the permission 'Full mailbox access'.		One of the standard permissions you can add to the mailbox.
Permission: Associated external account	Set this property to 'Yes' if you want to add the permission 'Associated external account'.		One of the standard permissions you can add to the mailbox. If you specify this permission, you must also specify permission <b>Full mailbox access</b> .
Use special permissions	Set this property to 'Yes' if you want to add a permission entry specified with the properties 'Special permission access mask', 'Special permission inheritance' and 'Special permission deny'.		Only use the special permissions if you cannot use the standard permissions. When you add a special permission, you also need to specify the properties: <b>Special permission access mask</b> and <b>Special permission inheritance</b> .
Special permission access mask	The access mask used for the access control entry that is added to the access control list of the mailbox. If you want to use special permissions, set property 'Use special permissions' to 'Yes'.		See <b>Use special permissions</b> .

Special permission inheritance	The inheritance settings used for the access control entry that is added to the access control list of the mailbox. If you want to use special permissions, set property 'Use special permissions' to 'Yes'.		See <b>Use special permissions</b> .
Permission deny flag	A flag indicating if the specified permission is granted or denied. Set to 'Yes' to deny access. When not specified or set to 'No', access is granted.		Set this flag to 'Yes' if the permission should be <b>denied</b> instead of <b>granted</b> . Normally you only specify permissions for a mailbox to grant access. You do not need to explicitly <b>deny</b> access to the mailbox.
Permission account is other account flag	A flag indicating if the permissions are updated for the account of the mailbox or another account. If set to 'Yes' a permission entry is added or removed for another account then the account of the mailbox. In this case you must also specify property 'Permission account name' or 'Permission account SID'.		You can add or remove permissions for the user account of the mailbox or another account. If you don't set this property to 'Yes', the specified permissions are updated for the account of the mailbox. If you want to update permissions for another account, you need to set this property to <b>Yes</b> and specify one of the following properties: <b>Permission account name</b> or <b>Permission account SID</b> to identify the other user account.

Permission account name	The name of an account for which an permission is added or permission are removed. If you want to use this property, you must also set the property 'Permission account is other account flag'.		See <b>Permission account is other account flag</b> .
Permission account SID	The security identifier (SID) of an account for which an permission is added or permission are removed. If you want to use this property, you must also set the property 'Permission account is other account flag'.		See <b>Permission account is other account flag</b> .
Remove account permission entries	A flag indicating if the permissions must be added or removed. If set to 'Yes', the permissions for the specified account (properties: 'Permission account is other account flag' and 'Permission account name' or 'Permission account SID') are removed from the mailbox access control list.		To remove permissions from the mailbox, set this flag to Yes. If another account is specified, the permissions for this account are removed from the mailbox. If no other account is specified, the explicit permissions for the account of the mailbox are removed.

**See also:**

*Script Action: Create Exchange Mailbox (2003/2000)* on page 156

*Script Action: Delete Exchange mailbox (2000/2003)* on page 168

*Script Action: Manage Exchange recipient mail addresses (2000/2003)* on page 169

*Script Action: Move Exchange mailbox* on page 167

*UMRA Basics* on page 3

*Script Action: Move Exchange mailbox*

### Function

Moves an existing Exchange 2003/2000 mailbox. The mailbox and user account must exist.

### Deployment

This action is typically used in a script that is intended to move the mailbox of existing user account. For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the *Script Action: Get user (AD)* on page 31. The **Get User** action supports several ways to find the user and fill the variable.

### Properties

Property Name	Description	Typical setting	Remarks
User Object	An data structure representing the user account. The property is used to identify the user account for the mailbox and is normally generated as a variable by a previous script action ('Creating user (AD)').	%UserObject%	This property specifies the mailbox that must exist (see <i>Script Action: Create Exchange Mailbox (2003/2000)</i> on page 156 for more information).
Mailbox destination	The object distinguished name of the destination mailbox store		

Domain controller	The (NETBIOS) name of the domain controller used to access the target Exchange mailbox store.	Optional	
-------------------	---	----------	--

**See also:**

*Script Action: Create Exchange Mailbox (2003/2000)* on page 156

*Script Action: Delete Exchange mailbox (2000/2003)* on page 168

*Script Action: Manage Exchange recipient mail addresses (2000/2003)* on page 169

*Script Action: Modify Exchange mailbox permissions* on page 162

*UMRA Basics* on page 3

**4.3.**

*Script Action: Delete Exchange mailbox (2000/2003)*

**Function**

Deletes the Exchange 2003 or Exchange 2000 mailbox of an existing user account. The user account is specified by a variable (default: %UserObject%). You can use the *Script Action: Get user (AD)* on page 31 to find the user account and initialize this variable.

**Deployment**

This action is typically used in a script that is intended to delete the mailbox of existing user account and possibly the user's resources and the account itself. For this action, the user account is identified by a variable (default: %UserObject%). To execute this action successfully, the variable must have a valid value. The variable is an output variable of the action *Script Action: Get user (AD)* on page 31. This action supports several ways to find the user and fill the variable.

**Properties**

Property Name	Description	Typical setting
User Object	A data structure representing the user account. The property is used to identify the user account for the mailbox and is normally generated as a variable by the <i>Script Action: Get user (AD)</i> on page 31.	%UserObject%

**See also:**

*Script Action: Create Exchange Mailbox (2003/2000)* on page 156

*Script Action: Manage Exchange recipient mail addresses (2000/2003)* on page 169

*Script Action: Modify Exchange mailbox permissions* on page 162

*Script Action: Move Exchange mailbox* on page 167

*UMRA Basics* on page 3

*Script Action: Manage Exchange recipient mail addresses (2003/2000)*

**Function**

Manages an Exchange mailbox for an Active Directory user account. This action supports MS Exchange versions 2003 and 2000.

**Deployment**

This action is typically used in a script that is intended to manage existing mailbox accounts.

**Properties**

Property Name	Description	Typical setting	Remarks
AD Object	A data structure representing the Active Directory object for which you want to manage the E-mail addresses.	%ActiveDirectoryObject%	This property is used to identify the mail recipient. You can obtain this variable by using the following script actions: Create user (AD), Create contact (AD), Get object (AD), Get user (AD). The output variable for these action must be set to %ActiveDirectoryObject%
Target address	The property specifies the delivery address to which e-mail for this recipient should sent. By specifying this property, mail is automatically enabled for the recipient.		If you specify this property, you should not specify the property 'Disable mail'.



Disable mail	With this property you can disable mail to a recipient. When set to 'Yes' the recipient can no longer receive mail and all mail addresses are cleared.	No	
Alias	Optional: The Alias property specifies the Alias used for E-mail address generation.		By default E-mail addresses are generated based on the name of the user account. The value is setup by MS Exchange automatically.

E-mail addresses	Optional: The explicit E-Mail addresses for the Exchange mail box.		<p>By default E-mail addresses are generated automatically when the mail box is created.</p> <p>Overruling of automatically generated addresses only occurs for the E-mail types that are explicitly set. That is, if your Exchange server configuration default generates both SMTP and X400 addresses, and the this property specifies only SMTP addresses, the X400 addresses will still be generated as specified on the Exchange server itself.</p> <p>Specify the E-mail address using the format (E-mail-type):(E-mail-Address). To specify the primary address, the E-mail-type must be in capitals. There must be exactly one primary E- mail address of each E-Mail type when used.</p> <p>Example:</p> <p>SMTP:J.Smith@tools4ever.com</p> <p>smtp:John@tools4ever.com</p>
------------------	---	--	--

Auto-update E-mail addresses	When this is set to 'Yes' Exchange will automatically generate E-mail addresses according to the Exchange recipient policy for the account.	Yes	
Hide from address book	When set to 'Yes', the user's mailbox does not show in address books.	No	
Restrict receiving message size	When set to 'Yes', messages larger than the specified maximum size will not be recieved.		
Maximum receiving message size	Specifies the maximum size, in kilobytes, of a messages that user or group can recieve.		

Restrict sending message size	When set to 'Yes', messages larger than the specified maximum size will not be send.		
Maximum sending message size	Specifies the maximum size, in kilobytes, of a messages that user or group can send.		

**See also:**

*Script Action: Create Exchange Mailbox (2003/2000) on page 156*

*Script Action: Delete Exchange mailbox (2000/2003) on page 168*

*Script Action: Modify Exchange mailbox permissions on page 162*

*Script Action: Move Exchange mailbox on page 167*

*UMRA Basics on page 3*

**Out-Of-Office**

Script Action: Get Out-Of-Office info (Exchange 2000/2003)

**Function**

Retrieves the Out-Of-Office and forwarding information of a user. This action can only be used for Exchange 2000 and Exchange 2003 environments. For Exchange 2007, use action 'Get Out-Of-Office info (Exchange 2007)'.

**Deployment**

This action is used to retrieve specific user related information from Exchange and store them in script variables, so that the information can be used by subsequent script actions.

**Special Prerequisites**

To execute this action, an Exchange profile must be configured for the account that runs the software. If an Exchange profile does not exist, the action will not work. For the UMRA Console and UMRA Service, different procedures apply:

**To configure an Exchange profile for the UMRA Console application:**

1. Log on to the computer that runs UMRA Console with an administrative account that has a mailbox on the Exchange server.
2. Install Microsoft Outlook (any version);
3. Start Outlook and setup a profile using Microsoft Exchange Server. A profile called 'Outlook' is now created.
4. Exit Outlook;
5. Start UMRA Console

**To configure an Exchange profile for the UMRA Service application:**

1. Create an Exchange mailbox for the UMRA Service account;
2. Stop the UMRA Service;
3. Log on the the computer that runs the UMRA Service with the account used by the UMRA Service (!). If you don't know the password of the account, first edit the password and update the password for the UMRA Service;
4. Install Microsoft Outlook (any version);
5. Start Outlook and setup a profile using Microsoft Exchange Server. A profile called 'Outlook' is now created.
6. Exit Outlook;
7. Start the UMRA Service.

### Properties

Property Name	Description	Typical setting	Remarks
Exchange Profile name	The name of the Exchange profile used to access the Exchange server. The Exchange profile must be the profile of the account that runs the software.	Outlook	See <b>Special Prerequisites</b>
User Object	An data structure representing the user account. The property is used to identify the user account and is normally generated as a variable by a previous script action (e.g. <i>Script Action: Get user (AD)</i> on page 31).	%UserObject%	

### Output Properties

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Out-Of-Office State	The Out-Of-Office state of the user account. The resulting output variable is either true (Out-Of-Office enabled) or false (Out-Of-Office disabled).	%OutOfOfficeState%	

Out-of-Office message	The Out-Of-Office message of the user account. The resulting output variable contains the message that is sent when Out-Of-Office is enabled and an E-mail message is received.	%OutOfOfficeText%	
Auto-Forward state	The Out-Of-Office Auto-Forward state of the user account. The resulting output variable is either true or false. When Out-Of-Office is enabled, received messages can be forwarded to another account (property value: 'true').	%AutoForwardState%	
Auto-Forward email address	The Out-Of-Office Auto-Forward email address of the user account. The resulting output variable contains the address of the E-mail account to which messages are forwarded when Out-Of-Office is enabled.	%AutoForwardEmail%	

**Remarks:**

The action will report an error when the user account does not have a mailbox or Exchange profile in the indicated Exchange environment

Script Action: Set Out-Of-Office info (Exchange 2000/2003)

### Function

Sets the Out-Of-Office and forwarding information of a user. This action can only be used for Exchange 2000 and Exchange 2003 environments. For Exchange 2007, use action 'Set Out-Of-Office info (Exchange 2007)'.

### Deployment

This action is used to turn on or off the Out-Of-Office functionality of a user. Optionally you can also specify whether incoming mail must be forwarded to another email address.

### Properties

Property Name	Description	Typical setting	Remarks
Exchange Profile name	The name of the Exchange profile used to access the Exchange server. The Exchange profile must be the profile of the account that runs the software.	Outlook	See <i>Script Action: Get Out-Of-Office info (Exchange 2000/2003)</i> on page 174 for more information.
User Object	An data structure representing the user account. The property is used to identify the user account and is normally generated as a variable by a previous script action (e.g. <i>Script Action: Get user (AD)</i> on page 31).	%UserObject%	



Out-Of-Office state	Specify 'Yes' to enable Out-Of-Office, 'No' to disable Out-Of-Office. Do not specify this property if the current value should not be changed.		
Out-of-Office message	The Out-Of-Office message that is used for the specified account.		This is the text that is sent by Exchange in a Out-Of-Office reply message.
Auto-Forward state	Specify 'Yes' to enable Auto-Forward, 'No' to disable. Do not specify this property if the current value should not be changed. When set to 'Yes', received messages will be forwarded to the specified E-mail address when Out-Of-Office is enabled.		
Auto-Forward E-mail address	The users' Auto-Forward E-mail address. When Out-Of-Office and Auto-Forward are enabled, received messages will be forwarded to the specified E-mail address.		

## Exchange 2007

### *User mailbox*

Script Action: Create user and mailbox (Exchange 2007)

### Function

Creates a new user in the Active Directory. The new user will be mailbox-enabled. This means a mailbox for this new user will be created on the specified Exchange server in the specified storage group.

### Deployment

This action is typically used as core part of a script designed to create users with a mailbox. In such a script this is usually the first major action invoked. After creating the account, the script usually continues by invoking actions to create home directories, home shares, group memberships, etc.

### Properties

Property Name	Description	Typical setting	Remarks
Database name	This parameter specifies which Exchange database will contain the new user's mailbox.	Mailbox Database EXCHSERVER/Mailbox Database EXCHSERVER/First Storage GroupMailbox Database	The Database parameter specifies a mailbox database on an Exchange server. If there are multiple Exchange server, make sure to include the servername in this parameter in order to get the right database. If there are multiple storage groups, specify the storage group as well in this parameter.

Name	<p>This parameter specifies the name of the new user. This is the name that appears in Active Directory Users and Computers as the common name. This is also the user name that appears in Recipient Properties on the User Information tab.</p>	John Smith	<p>The Name parameter specifies the name of the account. This is the same attribute as the 'Name' attribute of an account in Active Directory.</p>
------	--	------------	--

Password	<p>The password parameter specifies the initial password for the newly created user. Note that the password must meet password complexity requirements of the domain.</p>	test123	<p>To create the same password for all users you can specify the password here directly. You can also read the password from an input file.</p>
----------	---	---------	---

User principal name	<p>The 'User principal name' parameter specifies the user principal name (UPN) for this mailbox. This is the logon name for the user. The UPN consists of a user name and a suffix. Typically the suffix is the domain name where the user account resides.</p>	johnsmith@tools4ever.com	<p>The UPN is the preferred login name for Active Directory users. Users should be using their UPN to log on to the domain. The UPN has the format account_name@domain.com, where account_name is the UPN prefix and domain.com is the UPN suffix.</p> <p>The UPN Prefix is usually chosen to be the same as the SAM-Account-Name. Typically the name contained in %UserName% is generated by the name generation algorithm.</p>
---------------------	---	--------------------------	--

Organizational unit	The 'Organizational unit' parameter specifies the container where the user will be created. Specify the organizational unit with the domain name.	Tools4ever/Users	Specify the path of the organizational unit (OU) or container relative to the domain. To specify OU's in OU's, use the full path relative to the domain, separated by slashes: OU/ChildOU. Examples: Tools4ever/students or Tools4ever/students/group1.
---------------------	---	------------------	--

Alias	Optional value. The alias (mail nickname) of the user's new mailbox. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias. If not specified, the alias will be generated automatically.	JSmith	
-------	--	--------	--

Display name	Optional value. The display name of the new user created with this mailbox. The display name is the name that appears in the Exchange Management Console under Recipient Configuration. The Display Name also appears in Active Directory Users and Computers on the user Properties General Tab.	John Smith	If not set, the display name will be set with the 'Name' attribute of this action.
--------------	---	------------	--



Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	If a new account is created in a domain with multiple domain controllers, at first, the account will only be available at the domain controller where it is created. When the domain has synchronized all domain controllers in the domain, the account will be available at each domain controller. Until that time, if you want to edit a just created user account, you should specify the same domain controller in the 'Edit mailbox (Exchange 2007)' action.
First name	Optional value: The first name of the user account.	John	
Initials	Optional value: The initials of the user account.	F	

Last name	Optional value: The last name of the user account.	Smith	
Reset password on next logon	Optional value. If the 'Reset password on next logon' parameter is set to 'Yes', the user must change the password at the next logon.	'Yes' or 'No'	

SamAccount Name	Optional value. This parameter specifies the logon name used to support clients and servers running older versions of the operating system. This attribute must be less than 20 characters to support older clients. If not specified, Active Directory will create a SamAccount Name automatically, based on the user principal name.	JohnSmith	
-----------------	--	-----------	--

**Output Properties**

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Alias	The alias (mail nickname) of the user's new mailbox.		
Distinguished name	Optional value. This parameter returns the distinguished name of the new user mailbox.		
Exchange GUID	Optional value. This parameter returns the Exchange GUID of the new mailbox		
Legacy Exchange DN	Optional value. This parameter returns the Legacy Exchange Distinguished Name of the new mailbox.		

Script Action: Create (enable) mailbox (Exchange 2007)

### Function

Create a new Exchange 2007 mailbox for an existing Active Directory user account.

### Deployment

This action is typically used as core part of a script designed to create mailboxes for existing users. In such a script this is usually the first major action invoked. After creating the account, the script usually continues by invoking actions to create home directories, home shares, group memberships, etc.

### Properties

Property Name	Description	Typical setting	Remarks
Database name	This parameter specifies which Exchange database will contain the new mailbox for the user.	Mailbox database, SERVERNAME\Mailbox Database, SERVERNAME\Storage Group Name\Mailbox Database or the GUID of the database.	
Identity	This parameter specifies the user or InetOrgPerson of the new mailbox.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith@tools4ever.com or the GUID.	

Alias	Optional value. The alias (mail nickname) of the user's new mailbox. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias. If not specified, the alias will be generated automatically.	JSmith	
Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com.	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Alias	The alias (mail nickname) of the user's new mailbox.		
Exchange GUID	Optional value. This parameter returns the Exchange GUID of the new mailbox		
Legacy Exchange DN	Optional value. This parameter returns the Legacy Exchange Distinguished Name of the new mailbox.		

Script Action: Edit mailbox (Exchange2007)

### Function

Edit the attributes of an existing mailbox.

### Deployment

This action is typically used for editing the settings of mailboxes. Previous values of the properties listed below will be overwritten. Use \$null to clear properties.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mailbox.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
Accept messages only from	Optional value. The 'Accept messages only from' parameter specifies the mailbox users, mail users and mail contacts that can send e-mail messages to this mailbox. Use commas to specify multiple values.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JohnSmith, JSmith, John Smith, JSmith@tools4ever.com or the GUID.	



Alias	Optional value. The Alias parameter specifies the alias (mail nickname) of the user. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	
Anti spam bypass enabled	Optional value. The 'Anti spam bypass enabled' parameter specifies whether to skip anti-spam processing on this mailbox.	'Yes', 'No'	
Custom attribute 1	Optional value. The 'Custom attribute 1' parameter specifies the value for the mailbox attribute CustomAttribute1.		
Custom attribute 2	Optional value. The 'Custom attribute 2' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 3	Optional value. The 'Custom attribute 3' parameter specifies the value for the mailbox attribute CustomAttribute2.		

Custom attribute 4	Optional value. The 'Custom attribute 4' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 5	Optional value. The 'Custom attribute 5' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Deliver to mailbox and forward	Optional value. The 'Deliver to mailbox and forward' parameter specifies whether messages are sent to both this mailbox and the forwarding address.		
Display name	The 'Display name' parameter specifies the display name for the user account associated with this mailbox. The display name is used by Microsoft Outlook.	John Smith	
Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	

Email addresses	The 'Email addresses' parameter specifies all the proxy addresses of the mailbox. It includes the primary Simple Mail Transfer Protocol (SMTP) address as one of the proxy addresses. Use commas to specify multiple values.	SMTP:John@tools4ever.com, smtp:jsmith@tools4ever.com	
Emailaddress policy enabled	The 'Emailaddress policy enabled' parameter specifies whether the e-mail address policy for this mailbox is enabled.	'Yes' or 'No'	
External Out of Office options	The 'External Out of Office options' parameter specifies the option for sending an Out of Office message to external senders. Use one of the following values: 'External' and 'InternalOnly'.	External	

Forwarding address	The 'Forwarding address' parameter specifies a forwarding address. If 'Deliver to mailbox and forward' is set to 'Yes', messages that are sent to this mailbox will be forwarded to the address specified.	domain.com/Sales/JSmith	
Hidden from address lists enabled	The 'Hidden from address lists enabled' parameter specifies whether this mailbox is hidden from other address lists.	'Yes' or 'No'	
Issue warning quota	The 'Issue warning quota' parameter specifies the mailbox size at which a warning message is sent to the user. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited". If this attribute is set on a mailbox, it overrides the default value that is set for this attribute on the mailbox database.	50MB	

Maximum blocked senders	The 'Maximum blocked senders' parameter specifies the maximum number of senders that can be included in the blocked senders list. Blocked senders are senders that are considered junk senders by the mailbox user and are used in junk e-mail rules. This parameter is only validated when the junk e-mail rules are updated using Outlook Web Access or Web services.	5	
Maximum receive size	The 'Maximum receive size' parameter specifies the maximum size of messages that this mailbox can receive. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited".	50MB	

Maximum safe senders	The 'Maximum safe senders' parameter specifies the maximum number of senders that can be included in the safe senders list. Safe senders are senders that are trusted by the mailbox user and are used in junk e-mail rules. This parameter is only validated when the junk e-mail rules are updated using Outlook Web Access or Web services.	15	
Maximum send size	The 'Maximum send size' parameter specifies the maximum size of messages that this mailbox can send. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited".	50MB	
Name	The 'Name' parameter specifies the Name attribute for this mailbox. The Name attribute is used for the common name (CN) in Active Directory.	John Smith	

Office	The 'Office' parameter specifies the Microsoft Office attribute for this mailbox.		
Primary SMTP address	The 'Primary SMTP address' parameter specifies the address that external users will see when they receive a message from this mailbox. When this parameter is used the 'Email addresses' parameter cannot be specified because 'Email addresses' includes the primary SMTP address.	SMTP:jsmith@tools4ever.com	
Prohibit send quota	The 'Prohibit send quota' parameter specifies the mailbox size at which the user associated with this mailbox can no longer send messages. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited". If this attribute is set on a mailbox, it overrides the default value that is set for this attribute on the mailbox database.	50MB	

Prohibit send receive quota	The 'Prohibit send receive quota' parameter specifies the mailbox size at which the user associated with this mailbox can no longer send or receive messages. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited". If this attribute is set on a mailbox, it overrides the default value that is set for this attribute on the mailbox database.	50MB	
Recipient limits	The 'Recipient limits' parameter specifies the maximum number of recipients per message to which this mailbox can send. Specify either an integer or "unlimited". If this attribute is set on a mailbox, it overrides the default value that is set for this attribute on the hub transport server.	unlimited	



Reject messages from	The 'Reject messages from' parameter specifies the recipients from whom messages will be rejected. Use commas to specify multiple values.	domain.com/Sales/JSmith	
Require sender authentication enabled	The 'Require sender authentication enabled' parameter specifies whether senders must be authenticated.	'Yes' or 'No'	
Retain deleted items for	The 'Retain deleted items for' parameter specifies the length of time to keep deleted items. To specify a value, enter it as a time span: dd.hh:mm:ss where d = days, h = hours, m = minutes, and s = seconds.	15:00:00	
Retain deleted items until backup	The 'Retain deleted items until backup' parameter specifies whether to retain deleted items until the next backup.	'Yes' or 'No'	
Retention hold enabled	The 'Retention hold enabled' parameter specifies whether retention hold is enabled for MRM. To set the start date for retention hold, use the 'Start date for retention hold' parameter.	'Yes' or 'No'	

Rules quota	<p>The 'Rules quota' parameter specifies the limit for the size of rules for this mailbox. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes), TB (terabytes). Unqualified values are treated as bytes. The default value for this parameter is 64 KB. The maximum value for this parameter is 256 KB.</p>	128KB	
SamAccount Name	<p>The 'SamAccountName' parameter specifies the user name for earlier operating systems such as Windows NT 4.0, Windows 98, Windows 95, and LAN Manager. The parameter is used to support clients and servers running older versions of the operating system. This attribute must be less than 20 characters in length.</p>	jsmith	

Simple display name	The 'Simple display name' parameter is used on objects where the name of the object may be displayed in an environment that does not support Unicode characters. The only supported characters for the SimpleDisplayName parameter are ASCII characters 26 through 126, inclusively. These characters are the ones that are typically found on most U.S. English keyboards.	John Smith	
Start date for retention hold	The 'Start date for retention hold' parameter specifies the start date for retention hold for MRM. To use this parameter, the 'Retention hold enabled' parameter must be set to 'Yes'.	'Yes' or 'No'	

Use database quota defaults	The 'Use database quota defaults' parameter specifies that this mailbox uses the quota attributes specified for the mailbox database where this mailbox resides. The quota attributes are: 'ProhibitSendQuota', 'ProhibitSendReceiveQuota', 'IssueWarningQuota' and 'RulesQuota'.	'Yes' or 'No'	
Use database retention defaults	The 'Use database retention defaults' parameter specifies that this mailbox uses the MailboxRetention attribute specified for the mailbox database where this mailbox resides.	'Yes' or 'No'	
User principal name	The 'User principal name' parameter specifies the user principal name (UPN) for this mailbox. This is the logon name for the user. The UPN consists of a user name and a suffix. Typically, the suffix is the domain name where the user account resides.	jsmith@tools4ever.com	

Windows emailaddress	The 'Windows emailaddress' parameter specifies the Windows e-mail address for this mailbox. This address is not used by Exchange.	js@tools4ever.com	
----------------------	---	-------------------	--

Script Action: Manage mailbox email addresses (Exchange 2007)

#### Function

Manage the email addresses for a mailbox. Email addresses can be added or removed.

#### Deployment

This action is typically used to add or remove email addresses from the specified mailbox. Use an Umra variable for the Emailaddress property to add or remove that email address. If the Umra variable is a table or list, the action will read all items in the table or list and removes or adds these items to the mailbox. You can also edit the emailaddress property of a mailbox with the *Script Action: Edit mailbox (Exchange 2007)* on page 194 action. Note that this action overwrites the property, adding one emailaddress with this action clears the previous value of the property.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mailbox.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\jsmith', 'JSmith', 'JSmith@tools4ever.com' or the GUID.	

Emailaddress	The Emailaddress parameter specifies the proxy address of the mailbox. It includes the primary Simple Mail Transfer Protocol (SMTP) address as one of the proxy addresses. For example: smtp:JohnSmith@tools4ever.com". Use a table or text list to specify multiple values.	%EmailAddress%	
Remove address	With the 'Remove address' parameter the email address can be removed instead of added to the mailbox.	'Yes' or 'No'	

Script Action: Set client access attributes (Exchange 2007)

### Function

Set the client access-related attributes for Microsoft Exchange ActiveSync, Microsoft Office Outlook Web Access, Post Office Protocol version 3 (POP3), and Internet Message Access Protocol version 4rev1 (IMAP4) for a specified mailbox.

### Properties

Property Name	Description	Typical setting	Remarks

Identity	The Identity parameter identifies the mailbox. This can be the Active Directory Object ID or a string that represents the GUID, distinguished name, domain or account, user principal name (UPN), legacy Exchange distinguished name, or Simple Mail Transfer Protocol (SMTP) address.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\jsmith', 'jsmith', 'jsmith@tools4ever.com' or the GUID.	
----------	--	---	--



ActiveSyncAllowedDeviceIDs	This parameter accepts a list of device IDs that are allowed to synchronize with the mailbox.		
ActiveSyncDebugLogging	This parameter specifies whether error logging is enabled for mobile devices.		
ActiveSyncMailboxPolicy	This parameter specifies the name of the Exchange ActiveSync mailbox policy for the mailbox.		

ActiveSyncEnabled	This parameter enables or disables Exchange ActiveSync.		
HasActiveSyncDevicePartnership	This parameter specifies whether the mailbox has an active sync device partnership established.		

IgnoreDefaultScope	<p>This parameter instructs the command to ignore the default recipient scope setting for the Exchange Management Shell and to use the whole forest as the scope. This allows the command to access Active Directory objects that are currently not in the default scope.</p>		<p>This parameter can only be used when the Exchange Management Shell with SP1 is installed.</p> <p>Using this parameter introduces the following restrictions:</p> <p>You cannot use the DomainController parameter. The command will use an appropriate global catalog server automatically.</p> <p>You can only use the DN for the Identity parameter. Other forms of identification, such as alias or GUID are not accepted. You cannot use the</p>
--------------------	---	--	---

ImapEnabled			This parameter specifies whether the IMAP4 protocol is enabled for this mailbox.
-------------	--	--	--

ImapMessagesRetrievalMessageFormat	This parameter specifies the format of the messages that are retrieved from the server. The possible values are as follows: 0:Text Only 1:HTML Only 2:HTML and Alternative Text 3:Enriched Text Only 4:Enriched Text and Alternative Text 5:Best Body Format	3 or 0	
------------------------------------	--	--------	--

ImapUseProtocolDefaults	This parameter specifies whether to use protocol defaults for the IMAP4 protocol.		
MAPIBlockOutlookNonCachedMode	This parameter specifies whether Outlook can be used in online mode.		
MAPIBlockOutlookRpcHttp	This parameter specifies whether clients can connect to Outlook by using Outlook Anywhere.		

MAPIBlockOutlookVersions	This parameter specifies whether certain versions of Outlook are blocked.		
MAPIEnabled	This parameter specifies whether the MAPI protocol is enabled for the mailbox.		
OWAActiveSyncIntegration Enabled	This parameter specifies whether Outlook Web Access Exchange ActiveSync mobile options are enabled.		

OWAAllAddressListsEnabled	This parameter specifies whether all address lists are available in Outlook Web Access.		
OWACalendarEnabled	This parameter specifies whether calendar is enabled in Outlook Web Access.		
OWAChangePasswordEnabled	This parameter specifies whether a user can change their password in Outlook Web Access.		



OWAContactsEnabled	This parameter specifies whether contacts are enabled in Outlook Web Access.		
OWAEnabled	This parameter enables Outlook Web Access.		
OWAJournalEnabled	This parameter specifies whether the Journal folder can be accessed in Outlook Web Access.		

OWAJunkEmailEnabled	This parameter specifies whether management of junk e-mail is enabled in Outlook Web Access.		
OWANotesEnabled	This parameter specifies whether Sticky Notes are enabled in Outlook Web Access.		
OWAPremiumClientEnabled	This parameter specifies whether the Outlook Web Access Premium version is enabled.		

OWAPublicFoldersEnabled	This parameter specifies whether the viewing of public folders is enabled in Outlook Web Access.		This parameter can only be used when the Exchange Management Shell with SP1 is installed.
OWARecoverDeletedItemsEnabled	This parameter specifies whether recovery of deleted items is enabled in Outlook Web Access.		This parameter can only be used when the Exchange Management Shell with SP1 is installed.
OWARemindersAndNotificationsEnabled	This parameter specifies whether calendar reminders are enabled in Outlook Web Access.		

OWARulesEnabled	This parameter specifies whether rules can be accessed in Outlook Web Access. If this parameter is set to \$false, server rules will continue to function, but cannot be modified in Outlook Web Access.		This parameter can only be used when the Exchange Management Shell with SP1 is installed.
-----------------	--	--	---

OWASMimeEnabled	This parameter specifies whether viewing of e-mail that is encrypted by using S/MIME is supported in Outlook Web Access.		This parameter can only be used when the Exchange Management Shell with SP1 is installed.
OWASearchFoldersEnabled	This parameter specifies whether search folders are enabled in Outlook Web Access		

OWASignaturesEnabled	This parameter specifies whether the signature feature is enabled in Outlook Web Access.		
OWASpellCheckerEnabled	This parameter specifies whether the spelling checker is enabled in Outlook Web Access.		
OWATasksEnabled	This parameter specifies whether tasks are enabled in Outlook Web Access.		

OWAThemeSelectionEnabled	This parameter specifies whether theme selection is enabled in Outlook Web Access.		
OWAUMIntegrationEnabled	This parameter specifies whether Unified Messaging (UM) integration is enabled in Outlook Web Access.		

OWAUNCAccessOnPrivateComputersEnabled	This parameter specifies whether access to Windows file shares is permitted when users select This is a private computer on the Outlook Web Access logon page.		
---------------------------------------	--	--	--



OWAUNCAccessOnPublicComputersEnabled	This parameter specifies whether access to Windows file shares is permitted when users select This is a public or shared computer on the Outlook Web Access logon page.		
--------------------------------------	---	--	--

OWAWSSAccessOnPrivateComputersEnabled	This parameter specifies whether Windows SharePoint Services access is permitted when users select This is a private computer on the Outlook Web Access logon page.		
---------------------------------------	---	--	--

OWAWSSAccessOnPublicComputersEnabled	This parameter specifies whether Windows SharePoint Services access is permitted when users select This is a public or shared computer on the Outlook Web Access logon page.		
PopEnabled	This parameter specifies whether the POP3 protocol is enabled for a mailbox.		

PopMessagesRetrievalMimeFormat	This parameter specifies the format of the messages that are retrieved from the server. The possible values are as follows: 0:Text Only 1:HTML Only 2:HTML and Alternative Text 3:Enriched Text Only 4:Enriched Text and Alternative Text 6: Best Body Format	0 or 1	
--------------------------------	---	--------	--

PopUseProtocolDefaults	This parameter specifies whether to use protocol defaults for the POP3 protocol.		
ProtocolSettings	This parameter specifies the protocol settings.		
UseRusServer	This parameter instructs the command to use the specified RUS server to get and set mailbox and Active Directory user attributes.		

DomainController	This parameter specifies the fully qualified domain name (FQDN) of the domain controller that writes configuration changes to the Active Directory directory service.		
------------------	---	--	--

Script Action: Disable mailbox (Exchange 2007)

#### Function

Disable the mailbox of an existing user or InetOrgPerson. The user account associated with the mailbox will remain in Active Directory but will no longer be associated with a mailbox.

#### Deployment

This action disables a mailbox. The user account will remain in the Active Directory, but its mailbox properties will be removed. The mailbox will remain in the Exchange database. It will appear in the Exchange Management console under **Microsoft Exchange, Recipient Configuration, Disconnected Mailbox**.

With UMRA, use the action *Script Action: List mailbox statistics (Exchange 2007)* on page 326 and set the value of the **Disconnected mailboxes only** to 'Yes' to view all the disconnected mailboxes.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	This parameter specifies the mailbox you want to disable.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
Domain controller	The name of the domain controller to use to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	

Script Action: Remove user - mailbox (Exchange 2007)

### Function

Remove a user account that is associated with a particular mailbox in the Active Directory directory service and to process the associated, disconnected mailbox as directed by the specified parameters.

## Deployment

Use this action to remove the mailbox from the Exchange database and its user account from Active Directory. By default, the mailbox object will remain disconnected in the Exchange database for 30 days. To list all disconnected mailboxes use action *Script Action: List mailbox statistics (Exchange 2007)* on page 326 and set the value of the **Disconnected mailboxes only** to 'Yes'. To remove the mailbox object immediately, set the **Permanent** parameter of this action to 'Yes'.

## Properties

Property Name	Description	Typical setting	Remarks
Database name	This parameter specifies which Exchange database contains the mailbox object. This parameter must be used in conjunction with the StoreMailboxIdentity parameter. The Database parameter cannot be used with the Identity parameter.	'Mailbox database', 'SERVERNAME\Mailbox Database' or 'SERVERNAME\Storage Group Name\Mailbox Database'	



Identity	The Identity parameter identifies the mailbox object that you want to remove. The Identity parameter cannot be used with the Database parameter.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\Jsmith', 'JSmith', 'JSmith@tools4ever.com' or the GUID.	
Store mailbox identity	Optional value. The 'Store mailbox identity' parameter identifies the mailbox object to remove. The StoreMailboxIdentity parameter is used in conjunction with the Database parameter to remove the mailbox object from the Exchange database. To remove a disconnected mailbox from the Exchange store, use the Database and 'Store Mailbox Identity' parameters. Use the GUID of the mailbox to specify this parameter.	ca2c49e5-7536-49d0-83c5-bff5cd82e383	

Domain controller	The name of the domain controller to use to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	
Permanent	Optional value. The Permanent parameter, when used in conjunction with the Identity parameter, disconnects the mailbox from the user, removes the associated user object from Active Directory, and removes the mailbox object from the Exchange database. When set to 'No', the mailbox object remains in the Exchange database for 30 days, and then will be deleted. When the mailbox is empty it is always deleted immediately. The default value is 'No'.	'Yes' or 'No'	

Script Action: Connect mailbox (Exchange 2007)

**Function**

Connect a disconnected mailbox to an existing user object in the Active Directory.

**Properties**

Property Name	Description	Typical setting	Remarks
Database	The Database parameter specifies the Exchange database that contains the mailbox to connect a user to. If the server name is not specified in this parameter, this action will search for the database on the computer that runs the UMRA script.	Mailbox database, SERVERNAME\Mailbox Database, SERVERNAME\Storage Group Name\Mailbox Database or the GUID of the database.	

Identity	The Identity parameter specifies the mailbox object in the Exchange database to connect to an Active Directory user object. This parameter does not specify an Active Directory object.	/o=organization/ou=exchange admin group/cn=recipients/cn=jsmith, John Smith or the GUID of the database	
Alias	Optional value. The Alias parameter specifies the alias (mail nickname) for the mailbox after it is connected. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	

Domain controller	Optional value. The DomainController parameter specifies the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com.	
User	Optional value. The User parameter specifies the user object in Active Directory to connect the Exchange mailbox object to. If this parameter is not specified, the command will use the LegacyExchangeDN and DisplayName attributes of the Exchange mailbox object to find a user account that matches the mailbox object. If it cannot find a unique match, it will not connect the mailbox.	JSmith, John Smith, johnsmith@tools4ever.com	

Script Action: Move mailbox (Exchange 2007)

### Function

Move mailboxes within your organization or between different organizations.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mailbox.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
Target database	The 'Target database' parameter specifies the database to which the mailbox will be moved. If the server name is not specified, it searches the database on the computer that executes the UMRA script.	'Mailbox database', 'SERVERNAME\Mailbox Database' or 'SERVERNAME\Storage Group Name\Mailbox Database'.	

Allow merge parameter	Optional value. The 'Allow merge parameter' specifies the merging of mailboxes if one mailbox already exists. Use this parameter to move a mailbox between different organizations even if a target mailbox already exists. The contents of the mailbox are merged at the target. This parameter cannot be used if the NTAccountOU parameter is used.	'Yes' or 'No'	
-----------------------	---	---------------	--

Bad item limit	Optional value. The 'Bad item limit' parameter specifies the number of bad items to skip. Use 0 to not skip bad items. The valid input range for this parameter is 0 to 2,147,483,647.	15	
Configuration only	The 'Configuration only' parameter changes the Exchange server location in the Active Directory directory service. Use this parameter to direct the mailbox to a functional server. The mailbox content is not moved.	'Yes' or 'No'	



Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	
-------------------	---	-----------------------------	--

Script Action: Get mailbox permissions (Exchange 2007)

#### Function

Get the permissions of a mailbox. Multiple accounts can have access to a mailbox. This action generates a table with rows describing accounts and its access type.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mailbox.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	

User	Optional value: The User parameter specifies the user that has permissions on the mailbox. If not specified, the complete access rights list of the mailbox will be returned.	JSmith, John Smith, johnsmith@tools4ever.com or the GUID.	
Domain controller	Optional value. The name of the domain controller used to retrieve the data from Active Directory directory service. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	

**Output Properties**

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Mailboxpermissions	The resulting table with rows describing accounts and its access type on that mailbox.	%MailboxPermissions%	The available columns are: User, Accessrights, IsInherited, Deny and InheritanceType.

Script Action: Manage mailbox permissions (Exchange 2007)

#### Function

Manage the permissions of a mailbox by adding or removing access permissions for user accounts on a mailbox. Permission can also be removed for specific user accounts.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter specifies the identity of the mailbox that is getting permissions added or removed.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
User	Optional value. The User parameter specifies the user account that the permissions are being granted to or denied from on the other mailbox. If not specified, permissions are granted to or denied from the user that is implicitly associated with the mailbox by using NT AUTHORITY\SELF for the user.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com, 'John Smith' or the GUID.	

Access rights	The 'Access rights' specifies the target permission. Valid values are: FullAccess, SendAs, ExternalAccount, DeleteItem, ReadPermission, ChangePermission and ChangeOwner. If this parameter is not specified and the 'Remove permission' is set to 'Yes', all permissions of the user account will be removed. Use commas to specify multiple values.	ReadPermission, SendAs	
Deny	Optional value. If set to 'Yes', this parameter indicates that the specified access rights are to be added to the list of rights that are explicitly denied to the indicated user. The default value is 'No'.	'Yes' or 'No'	

Remove permission	If set to 'Yes' this parameter specifies that the right is to be removed from the list of rights instead of added. If the 'Deny' parameter is also specified, the right is removed from the list of rights that are explicitly denied. The default value is 'No'.	'Yes' or 'No'	
Domain controller	Optional value. The name of the domain controller used to write this configuration update to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com.	

Inheritance type	Optional value. Use this parameter to specify whether permissions are inherited to folders within the mailbox. Valid values are: None, All, Descendants, SelfAndChildren, Children.	SelfAndChildren	
------------------	---	-----------------	--

Script Action: List mailboxes (Exchange 2007)

#### Function

Retrieve information about mailboxes. The action returns a table with rows for each mailbox.

#### Deployment

Use this action to get an overview of mailboxes in Exchange. This action provides three **output properties**. The simple, regular and advanced table. The simple table contains the most important values of a mailbox. The advanced table contains all properties, but therefore causes more network traffic.

#### Properties

Property Name	Description	Typical setting	Remarks
Database	Optional value. The Database parameter specifies the database from which to get the mailbox. If the server name is not specified in this parameter, this action will search for the database on the computer that runs the UMRA script. This parameter cannot be used in conjunction with the Filter parameter.	'Mailbox database', 'SERVERNAME\Mailbox Database', 'SERVERNAME\Storage Group Name\Mailbox Database'	



Organization al unit	Optional value. The 'Organization al unit' parameter specifies an organization al unit (OU), and is used to limit the results. If this parameter is specified, only mailboxes in the specified container will be retrieved. Use either the OU or the domain name. If the OU is used, specify the canonical name of the OU.	'Tools4ever.com/Management/Sales'	
-------------------------	---	-----------------------------------	--

Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com	
Filter	Optional value. The Filter parameter specifies a set of attributes that restricts the mailboxes that are returned by the query. This parameter cannot be used in conjunction with the 'Database' parameter.	Name -like '*Smith'	

Result size	Optional value. The ResultSize parameter sets the maximum number of results to return. If all mailboxes should be returned, use "unlimited" for the value of this parameter. The default value is 1000.	1000	
Identity	The Identity parameter identifies the mailbox. If this parameter is specified only the mailbox associated with the particular mailbox user is returned.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\JSmith, JSmith jsmith@tools4ever.com or the GUID	
Sort by	Optional value. The SortBy parameter sorts by a single attribute in ascending order.	Name	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
UserMailboxSimple	The resulting table with columns for Name, SamAccountName and DistinguishedName for each mailbox.		
UserMailboxRegular	The resulting table with columns for Name, Alias, ServerName, DisplayName, SamAccountName, UserPrincipalName, PrimarySmtpAddress and DistinguishedName for each mailbox.	%UserMailboxRegular%	

UserMailboxAdvanced	The resulting table with many columns for each mailbox.		The columns available are: AcceptMessagesOnlyFrom, Alias, AntiSpamBypassEnabled, CustomAttribute1, CustomAttribute2, CustomAttribute3, CustomAttribute4, CustomAttribute5, DeliverToMailboxAndForward, DisplayName, DistinguishedName, EmailAddresses, EmailAddressPolicyEnabled, ExternalOofOptions, ForwardingAddress, HiddenFromAddressListsEnabled, IssueWarningQuota, MaxBlockedSenders, MaxReceiveSize, MaxSafeSenders, MaxSendSize, Name, Office, PrimarySmtpAddress, ProhibitSendQuota, ProhibitSendReceiveQuota, RecipientLimits, RejectMessagesFrom, RequireSenderAuthenticationEnabled, RetainDeletedItemsFor, RetainDeletedItemsUntilBackup, RetentionHoldEnabled, RulesQuota, SamAccountName, SimpleDisplayName, StartDateForRetentionHold, UseDatabaseQuotaDefaults, UseDatabaseRetentionDefaults, UserPrincipalName, WindowsEmailAddress.
---------------------	---	--	--

*Mail user*

Script Action: Create mail user (Exchange 2007)

**Function**

Create a new mail user in the Active Directory directory service. A mail user has no mailbox, but can receive mail. The mail received will be sent to his 'External emailaddress'.

**Properties**

Property Name	Description	Typical setting	Remarks
External emailaddress	The 'External emailaddress' parameter specifies an e-mail address outside of the organization. E-mail messages sent to the mail-enabled user are sent to this external address.	johnsmith@external.com	
Name	The Name parameter specifies the common name (CN) of the mail-enabled user.	John Smith	
User principal name	The 'User principal name' parameter defines the name of a system user in an e-mail address format.	johnsmith@tools4ever.com	

Password	The password parameter specifies the initial password for the newly created mail user. Note that the password must meet password complexity requirements as set for the domain.	test123	
Organizational unit	The 'Organizational unit' parameter specifies the organizational unit in which the new mail user is added.	tools4ever.com/Sales	
Alias	Optional value. The email alias of the mail user. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	

Display name	Optional value. The 'Display name' parameter specifies the name that will be displayed in Microsoft Outlook for the mail user.	John Smith	
First name	Optional value. The first name of the mail user.	John	
Initials	Optional value. The initials of the mail user.	F	
Last name	Optional value. The last name of the mail user.	Smith	
SamAccountName	Optional value. This parameter specifies the logon name used to support clients and servers running older versions of the operating system. This attribute must be less than 20 characters to support older clients. If this parameter is not specified, Active Directory will create a SamAccountName automatically, based on the user principal name.	JohnSmith	



Reset password on next logon	Optional value. If the 'Reset password on next logon' parameter is set to 'Yes', the user must change the password at the next logon.	'Yes', 'No'	
Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com. .	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Alias	Optional value. The alias of the mail contact.		

SamAccountName	Optional value. This parameter specifies the logon name used to support clients and servers running older versions of the operating system.		
Distinguished name	Optional value. This parameter returns the distinguished name of the new mail user.		
GUID	Optional value. This parameter returns the GUID of the new mail user.		

Script Action: Enable mail user (Exchange 2007)

#### **Function**

Mail enable an existing user account in the Active Directory. A mail user has no mailbox, but can receive mail. The mail received will be sent to his 'External emailaddress'.

**Properties**

Property Name	Description	Typical setting	Remarks
External emailaddress	The 'External emailaddresses' parameter specifies an e-mail address outside of the organization. E-mail messages sent to the mail-enabled user is sent to this external address.	johnsmith@external.com	
Identity	This parameter specifies the user that will be mail-enabled.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\jsmith, JSmith@tools4ever.com or the GUID.	

Alias	Optional value. The email alias of the user. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	
Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com.	

**Output Properties**

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Alias	Optional value. The email alias of the user.		
GUID	Optional value. This parameter returns the GUID of the new mail user.		
Legacy Exchange DN	Optional value. This parameter returns the Legacy Exchange DN of the new mail user.		

Script Action: Edit mail user (Exchange 2007)

#### Function

Edit the attributes of a mail enabled user by specifying the properties.

### Deployment

This action is typically used for editing the settings of mail users.

Previous values of the properties listed below will be overwritten. Use \$null to clear properties.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mail enabled user.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
Accept messages only from	Optional value. The 'Accept messages only from' parameter specifies the mailbox users, mail users and mail contacts that can send e-mail messages to this mailbox. Use commas to specify multiple values.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JohnSmith, JSmith, John Smith, JSmith@tools4ever.com or the GUID.	

Alias	Optional value. The Alias parameter specifies the alias of the mail enabled user. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	
Custom attribute 1	Optional value. The 'Custom attribute 1' parameter specifies the value for the mailbox attribute CustomAttribute1.		
Custom attribute 2	Optional value. The 'Custom attribute 2' parameter specifies the value for the mailbox attribute CustomAttribute2.		

Custom attribute 3	Optional value. The 'Custom attribute 3' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 4	Optional value. The 'Custom attribute 4' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 5	Optional value. The 'Custom attribute 5' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Display name	The 'Display name' parameter specifies the display name of the user.	John Smith	



Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	
Email addresses	The 'Email addresses' parameter can be used to specify the e-mail alias of the mail user. All valid Microsoft Exchange 2007 e-mail address types may be used. Use commas to specify multiple values.	SMTP:John@tools4ever.com, smtp:jsmith@tools4ever.com	
Emailaddress policy enabled	The 'Emailaddress policy enabled' parameter specifies whether the e-mail address policy for this mailbox is enabled.	'Yes' or 'No'	

External email address	The 'External email address' parameter specifies whether the e-mail addresses for the mailbox will be automatically updated based on the e-mail address policies defined. When this parameter is set to 'Yes', the 'Primary SMTP address' or 'Windows email address' parameters cannot be changed.	smtp:jsmith@tools4ever.com	
Grant send on behalf to	The 'Grant send on behalf to' parameter specifies the distinguished name (DN) of recipients that can send messages on behalf of this user.	CN=jsmith,OU=sales,DC=tools4ever,DC=com	
Hidden from address lists enabled	The 'Hidden from address lists enabled' parameter specifies whether the user appears in the address lists.	'Yes' or 'No'	

Maximum receive size	The 'Maximum receive size' parameter specifies the maximum size of e-mail messages that can be received by the mail user, from 1 kilobyte (KB) to 2,097,151 KB. If not specified, there will be no size restrictions. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited".	50MB	
----------------------	--	------	--

Maximum send size	The 'Maximum send size' parameter specifies the maximum size of e-mail messages that can be sent by the mail user, from 1 KB to 2,097,151 KB. If not specified, there will be no size restrictions. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited".	50MB	
Name	The 'Name' parameter specifies the name of the user.	John Smith	

Primary SMTP address	The 'Primary SMTP address' parameter specifies the address that external users will see when they receive a message from this mailbox. If this parameter is used, the 'Email addresses' parameter cannot be used because 'Email addresses' includes the primary SMTP address.	SMTP:jsmith@tools4ever.com	
Recipient limits	The 'Recipient limits' parameter specifies the maximum number of recipients for messages from this user. Specify either an integer or "unlimited." If this attribute is set on a mailbox, it overrides the default value that is set for this attribute on the hub transport server.	unlimited	

Reject messages from	The 'Reject messages from' parameter specifies the recipients from whom messages will be rejected. Use commas to specify multiple values.	domain.com/Sales/JSmith	
Require sender authentication enabled	The 'Require sender authentication enabled' parameter specifies whether senders must be authenticated.	'Yes' or 'No'	

SamAccountName	The 'SamAccountName' parameter defines the logon name used to support clients and servers running older versions of the operating system, such as Microsoft Windows NT 4.0, Windows 98, Windows 95, and LAN Manager. This parameter must be less than 20 characters in length.	jsmith	
----------------	--	--------	--

Simple display name	The 'Simple display name' parameter is used on objects where the name of the object may be displayed in an environment that does not support Unicode characters. The only supported characters for the SimpleDisplayName parameter are ASCII characters 26 through 126, inclusively. These characters are the ones that are typically found on most U.S. English keyboards.	John Smith	
User principal name	The 'User principal name' parameter specifies the user principal name (UPN) for the mail user.	johnsmith@tools4ever.com	



Windows emailaddress	The 'Windows emailaddress' parameter specifies the Windows e-mail address for this mail user. This address is not used by Exchange.	js@tools4ever.com	
----------------------	---	-------------------	--

Script Action: Manage mail user email addresses (Exchange 2007)

### Function

Manage the additional email addresses for a mail user. Email addresses can be added or removed.

### Deployment

This action is typically used to add or remove email addresses from the specified mail enabled user. Use an Umra variable for the Emailaddress property to add or remove that email address. If the Umra variable is a table or list, the action will read all items in the table or list and removes or adds these items to the mailbox. You can also edit the emailaddress property of a mail user with the *Script Action: Edit mail user (Exchange 2007)* on page 263 action. Note that this action overwrites the property, adding one emailaddress with this action clears the previous value of the property.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mail enabled user.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\Jsmith', 'JSmith', 'JSmith@tools4ever.com' or the GUID.	
Emailaddresses	The Emailaddresses parameter specifies the proxy address of the mail enabled user. It includes the primary Simple Mail Transfer Protocol (SMTP) address as one of the proxy addresses. Use a table or text list to specify multiple values.	%EmailAddress%, 'smtp:JohnSmith@tools4ever.com'	

Remove address	With the 'Remove address' parameter the email address can be removed instead of added to the mail enabled user.	'Yes' or 'No'	
----------------	---	---------------	--

Script Action: Disable mail user (Exchange 2007)

#### Function

Disable a mail-enabled user and remove that object's Exchange attributes from the Active Directory directory service.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	This parameter specifies the user to mail-disable.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\jsmith, JSmith, JSmith@tools4ever.com or the GUID.	

Domain controller	The name of the domain controller to use to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	
-------------------	---	-----------------------------	--

Script Action: Remove mail user (Exchange 2007)

#### Function

Remove a mail enabled user from the Active Directory directory service.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mail enabled user.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\jsmith', 'jsmith', 'jsmith@tools4ever.com' or the GUID.	

Domain controller	The name of the domain controller to use to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	
-------------------	---	-----------------------------	--

Script Action: List mail users (Exchange 2007)

### Function

Retrieve information about mail users. The action returns a table with rows for each mail user.

### Deployment

Use this action to get an overview of mail users in Exchange. This action provides three **output properties**. The simple, regular and advanced table. The simple table contains the most important values of a mailbox. The advanced table contains all properties, but therefore causes more network traffic.

### Properties

Property Name	Description	Typical setting	Remarks
Organizational unit	Optional value. The 'Organizational unit' parameter specifies an organizational unit (OU), and is used to limit the results. Use this parameter to return only the mail enabled users in the specified container. Specify either the OU or the domain name.	Tools4ever.com/Management/Sales	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com	

Filter	Optional value. The Filter parameter specifies a set of attributes that restricts the results that are returned by the query.	Name -like '*Smith'	
Result size	Optional value. The ResultSize parameter specifies the maximum number of results to return. To return all mail enabled users that match the query, use "unlimited" for the value of this parameter. The default value is 1000.	1000	
Identity	The Identity parameter identifies the mail enabled user.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\JSmith, JSmith jsmith@tools4ever.com or the GUID	

Sort by	Optional value. The SortBy parameter sorts by a single attribute in ascending order.	Name	
---------	--	------	--

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
MailUserSimple	The resulting table with columns for Name, SamAccountName and Distinguished Name for each user.		



MailUserRegular	The resulting table with columns for Name, Alias, DisplayName, SamAccountName, UserPrincipalName, PrimarySmtpAddress and Distinguished Name for each mail user.	%MailUserRegular%	
-----------------	---	-------------------	--

MailUserAdvanced	The resulting table with many columns for each mail user.		The columns available are: AcceptMessagesOnlyFrom, AddressListMembership, Alias, CustomAttribute1, CustomAttribute2, CustomAttribute3, CustomAttribute4, CustomAttribute5, DisplayName, DistinguishedName, EmailAddresses, EmailAddressPolicyEnabled, ExchangeUserAccountControl, ExchangeVersion, Extensions, ExternalEmailAddress, GrantSendOnBehalfTo, Guid, HiddenFromAddressListsEnabled, IsValid, LegacyExchangeDN, MacAttachmentFormat, MaxReceiveSize, MaxSendSize, MessageBodyFormat, MessageFormat, Name ObjectCategory, ObjectClass, OrganizationalUnit, OriginatingServer, PoliciesExcluded, PoliciesIncluded, PrimarySmtpAddress, ProtocolSettings, RecipientLimits, RecipientType, RecipientTypeDetails, RejectMessagesFrom, RejectMessagesFromDLMembers, RequireSenderAuthenticationEnabled, SamAccountName, SimpleDisplayName, UMDtmfMap, UseMapiRichTextFormat, UsePreferMessageFormat, UserPrincipalName, WhenChanged, WhenCreated, WindowsEmailAddress
------------------	---	--	---

*Mail contact*

Script Action: Create mail contact (Exchange 2007)

**Function**

Create a new mail enabled contact. A new contact object will be created in the Active Directory. This new contact will be mail enabled. Mail received by this contact will be sent to its 'External emailaddress'.

**Deployment****Properties**

Property Name	Description	Typical setting	Remarks
External emailaddress	The 'External emailaddress' parameter specifies an e-mail address outside of the organization. E-mail messages sent to the mail-enabled contact are sent to this external address.	johnsmith@external.com	
Name	The Name parameter specifies the common name (CN) of the mail contact.	John Smith	

Organizational unit	The 'Organizational unit' parameter specifies the container where the new contact will be created. Specify the organizational unit with the domain name.	tools4ever.com/Sales	
Alias	Optional value. The alias of the mail contact. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	
Display name	Optional value. The 'Display name' parameter specifies the name that will be displayed in Microsoft Outlook for the mail contact.	John Smith	

First name	Optional value. The first name of the mail contact.	John	
Initials	Optional value. The initials of the mail contact.	F	
Last name	Optional value. The last name of the mail contact.	Smith	
Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Alias	Optional value. The alias of the mail contact.		
Distinguished name	Optional value. This parameter returns the distinguished name of the new mail contact.		
GUID	Optional value. This parameter returns the GUID of the new mail contact.		

Script Action: Enable mail contact (Exchange 2007)

#### Function

Mail enable an existing contact in the Active Directory. Mail received by this contact will be sent to its 'External emailaddress'.

#### Deployment

#### Properties

Property Name	Description	Typical setting	Remarks
External emailaddress	The 'External emailaddresses' parameter specifies an e-mail address outside of the organization. E-mail messages sent to the mail-enabled contact is sent to this external address.	johnsmith@external.com	
Identity	This parameter specifies the contact that will be mail-enabled.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith@tools4ever.com or the GUID.	

Alias	Optional value. The email alias of the contact. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	
Display name	Optional value. The 'Display name' of the mail contact.	John Smith	



Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com.	
-------------------	---	----------------------------	--

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Alias	Optional value. The email alias of the contact.		

GUID	Optional value. This parameter returns the GUID of the new mail contact.		
------	--	--	--

Script Action: Edit mail contact (Exchange 2007)

#### Function

Edit the attributes of a mail enabled contact.

#### Deployment

This action is typically used for editing the settings of mail contacts. Previous values of the properties listed below will be overwritten. Use \$null to clear properties.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mail enabled contact.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	

Accept messages only from	Optional value. The 'Accept messages only from' parameter specifies the mailbox users, mail users and mail contacts that can send e-mail messages to this mailbox. Use commas to specify multiple values.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JohnSmith, JSmith, John Smith, JSmith@tools4ever.com or the GUID.	
Alias	Optional value. The Alias parameter specifies the alias of the mail enabled contact. The alias can be a combination of characters separated by a period with no intervening spaces. Do not use special characters in the alias.	JSmith	
Custom attribute 1	Optional value. The 'Custom attribute 1' parameter specifies the value for the mailbox attribute CustomAttribute1.		

Custom attribute 2	Optional value. The 'Custom attribute 2' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 3	Optional value. The 'Custom attribute 3' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 4	Optional value. The 'Custom attribute 4' parameter specifies the value for the mailbox attribute CustomAttribute2.		
Custom attribute 5	Optional value. The 'Custom attribute 5' parameter specifies the value for the mailbox attribute CustomAttribute2.		

Display name	The 'Display name' parameter specifies the display name of the contact.	John Smith	
Domain controller	Optional value. The name of the domain controller used to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	
Email addresses	The 'Email addresses' parameter can be used to specify the e-mail alias of the mail contact. All valid Microsoft Exchange 2007 e-mail address types may be used. Use commas to specify multiple values.	SMTP:John@tools4ever.com, smtp:jsmith@tools4ever.com	

Emailaddress policy enabled	The 'Emailaddress policy enabled' parameter specifies whether the e-mail addresses for the mailbox will be automatically updated based on the e-mail address policies defined.	'Yes' or 'No'	
External email address	The 'External email address' parameter specifies whether the e-mail addresses for the mailbox will be automatically updated based on the e-mail address policies defined. When this parameter is set to 'Yes', the 'Primary SMTP address' or 'Windows email address' parameters cannot be changed.	smtp:jsmith@tools4ever.com	

Grant send on behalf to	The 'Grant send on behalf to' parameter specifies the distinguished name (DN) of recipients that can send messages on behalf of this contact.	CN=jsmith,OU=sales,DC=tools4ever,DC=com	
Hidden from address lists enabled	The 'Hidden from address lists enabled' parameter specifies whether the contact appears in the address lists.	'Yes' or 'No'	
Maximum receive size	The 'Maximum receive size' parameter specifies the maximum size of e-mail messages that can be received, from 1 kilobyte (KB) to 2,097,151 KB. If not specified, there will be no size restrictions. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited".	50MB	

Maximum recipient per message	The 'Maximum recipient per message' parameter specifies the maximum number of recipients for messages from this mail contact.	50	
Maximum send size	The 'Maximum send size' parameter specifies the maximum size of e-mail messages that can be sent, from 1 KB to 2,097,151 KB. If not specified, there will be no size restrictions. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes) or "unlimited".	50MB	
Name	The 'Name' parameter specifies the name of the contact.	John Smith	



Primary SMTP address	The 'Primary SMTP address' parameter specifies the address that external users will see when they receive a message from this mailbox. If this parameter is used, the 'Email addresses' parameter cannot be used because 'Email addresses' includes the primary SMTP address.	SMTP:jsmith@tools4ever.com	
Reject messages from	The 'Reject messages from' parameter specifies the recipients from whom messages will be rejected. Use commas to specify multiple values.	domain.com/Sales/JSmith	
Require sender authentication enabled	The 'Require sender authentication enabled' parameter specifies whether senders must be authenticated.	'Yes' or 'No'	

Simple display name	The 'Simple display name' parameter is used on objects where the name of the object may be displayed in an environment that does not support Unicode characters. The only supported characters for the SimpleDisplayName parameter are ASCII characters 26 through 126, inclusively. These characters are the ones that are typically found on most U.S. English keyboards.	John Smith	
Windows emailaddress	The 'Windows emailaddress' parameter specifies the Windows e-mail address for this mail contact. This address is not used by Exchange.	js@tools4ever.com	

Script Action: Manage mail contact email addresses (Exchange 2007)

### Function

Manage the additional email addresses for a mail contact. Email addresses can be added or removed.

### Deployment

This action is typically used to add or remove email addresses from the specified mail contact. Use an Umra variable for the Emailaddress property to add or remove that email address. If the Umra variable is a table or list, the action will read all items in the table or list and removes or adds these items to the mailbox. You can also edit the emailaddress property of a mail contact with the *Script Action: Edit mail contact (Exchange 2007)* action. Note that this action overwrites the property, adding one emailaddress with this action clears the previous value of the property.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mail contact.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\Jsmith', 'JSmith', 'JSmith@tools4ever.com' or the GUID.	

Emailaddress	The Emailaddress parameter specifies all the proxy addresses of the mail contact. It includes the primary Simple Mail Transfer Protocol (SMTP) address as one of the proxy addresses.	%EmailAddress%, 'smtp:JohnSmith@tools4ever.com'	
Remove address	With the 'Remove address' parameter the email address can be removed instead of added to the mail contact.	'Yes' or 'No'	

Script Action: Disable mail contact (Exchange 2007)

#### Function

Disable a mail-enabled contact and remove that object's Exchange attributes from the Active Directory directory service.

**Deployment****Properties**

Property Name	Description	Typical setting	Remarks
Identity	This parameter specifies the contact to mail-disable.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
Domain controller	The name of the domain controller to use to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	

Script Action: Remove mail contact (Exchange 2007)

**Function**

Remove a mail enabled contact from the Active Directory directory service.

## Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the mail enabled contact.	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\Jsmith', 'JSmith', 'JSmith@tools4ever.com' or the GUID.	
Domain controller	The name of the domain controller to use to write this configuration change to Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com..	

Script Action: List mail contacts (Exchange 2007)

## Function

Retrieve information about mail contacts. The action returns a table with rows for each mail contact.

## Deployment

Use this action to get an overview of mail contacts in Exchange. This action provides three **output properties**. The simple, regular and advanced table. The simple table contains the most important values of a mail contact. The advanced table contains all properties, but therefore causes more network traffic.

**Properties**

Property Name	Description	Typical setting	Remarks
Organizational unit	Optional value. The 'Organizational unit' parameter specifies an organizational unit (OU), and is used to limit the results. Use this parameter to return only the mail enabled contacts in the specified container. Specify either the OU or the domain name.	Tools4ever.com/Management/Sales	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com	

Filter	Optional value. The Filter parameter specifies a set of attributes that restricts the results that are returned by the query.	Name -like '*Smith'	
Recipient type details	Optional value. The 'Recipient type details' parameter specifies the type of recipients that are returned. For this action, the available recipient type details are: MailContact or MailForestContact.	MailContact	
Result size	Optional value. The ResultSize parameter specifies the maximum number of results to return. To return all mail contacts that match the query, use "unlimited" for the value of this parameter. The default value is 1000.	1000	



Identity	The Identity parameter identifies the mail contact.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\JSmith, JSmith jsmith@tools4ever.com or the GUID	
Sort by	Optional value. The SortBy parameter sorts by a single attribute in ascending order.	Name	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
MailContactSimple	The resulting table with columns for Name, DisplayName and DistinguishedName for each contact.		

MailContactRegular	The resulting table with columns for Name, DisplayName and DistinguishedName, Alias, OrganizationalUnit, SimpleDisplayName for each mail contact.	%MailContactRegular%	
--------------------	---	----------------------	--

MailContactAdvanced	The resulting table with many columns for each mail contact.		The columns available are: AcceptMessagesOnlyFrom, AcceptMessagesOnlyFromDLMembers, AddressListMembership, Alias, CustomAttribute1, CustomAttribute2, CustomAttribute3, CustomAttribute4, CustomAttribute5, DisplayName, DistinguishedName, EmailAddresses, EmailAddressPolicyEnabled, ExchangeVersion, Extensions, ExternalEmailAddress, GrantSendOnBehalfTo, Guid, HiddenFromAddressListsEnabled, IsValid, LegacyExchangeDN, MacAttachmentFormat, MaxReceiveSize, MaxRecipientPerMessage, MaxSendSize, MessageBodyFormat, MessageFormat, Name, ObjectCategory, ObjectClass, OrganizationalUnit, OriginatingServer, PoliciesExcluded, PoliciesIncluded, PrimarySmtpAddress, RecipientType, RecipientTypeDetails, RejectMessagesFrom, RejectMessagesFromDLMembers, RequireSenderAuthenticationEnabled, SimpleDisplayName, UMDtmfMap, UseMapiRichTextFormat, UsePreferMessageFormat, WhenChanged, WhenCreated, WindowsEmailAddress
---------------------	--	--	---

*User*

Script Action: List users (Exchange 2007)

**Function**

Retrieve all users in the forest that match the specified conditions. All results are returned in a table with one row for each user account.

Various return tables are available with different columns.

**Deployment**

Use this action to get an overview of users. This action provides three **output properties**. The simple, regular and advanced table. The simple table contains the most important values of a user. The advanced table contains all properties, but therefore causes more network traffic.

**Properties**

Property Name	Description	Typical setting	Remarks
Organization al unit	Optional value. The 'Organization al unit' parameter returns users only from the specified organization al unit (OU).	Tools4ever.com/Management/Sales	

Filter	Optional value. The Filter parameter specifies a set of attributes that restricts the users that are returned by the query.	Title -like '*Manager'	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com	

Result size	Optional value. The ResultSize parameter sets the maximum number of results to return. If all users should be returned, use "unlimited" for the value of this parameter. The default value is 1000.	1000	
Identity	The Identity parameter identifies the users. If this parameter is specified, only the specified user is returned.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\JSmith, JSmith jsmith@tools4ever.com or the GUID	
Sort by	Optional value. The SortBy parameter sorts by a single attribute in ascending order.	Name	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
UserTableSimple	The resulting table with columns for Name, SamAccountName and DistinguishedName for each user.		
UserTableRegular	The resulting table with columns for Name, SamAccountName, DistinguishedName, RecipientType, UserPrincipalName and Sid for each user.	%UserTableRegular%	

UserTableAdvanced	The resulting table with many columns for each user.		The columns available are:  AssistantName, City, Company, CountryOrRegion, Department, DirectReports, DisplayName, DistinguishedName, ExchangeVersion, Fax, FirstName, Guid, HomePhone, Identity, Initials, IsSecurityPrincipal, IsValid, LastName, Manager, MobilePhone, Name, Notes, ObjectCategory, ObjectClass, Office, OriginatingServer, OtherFax, OtherHomePhone, OtherTelephone, Pager, Phone, PhoneticDisplayName, PostalCode, PostOfficeBox, RecipientType, RecipientTypeDetails, ResetPasswordOnNextLogon, SamAccountName, Sid, SidHistory, SimpleDisplayName, StateOrProvince, StreetAddress, Title, UMDialPlan, UMDtmfMap, UserPrincipalName, WebPage, WhenChanged, WhenCreated, WindowsEmailAddress.
-------------------	--	--	--



### *Contact*

Script Action: List contacts (Exchange 2007)

#### **Function**

Retrieve all contacts from Active Directory that match the specified conditions. All results are returned in a table with one row for each contacts. Various return tables are available with different columns.

#### **Deployment**

Use this action to get an overview of contacts. This action provides three **output properties**. The simple, regular and advanced table. The simple table contains the most important values of a contact. The advanced table contains all properties, but therefore causes more network traffic.

#### **Properties**

Property Name	Description	Typical setting	Remarks
Organizational unit	Optional value. The 'Organizational unit' parameter returns contacts only from the specified organizational unit (OU).	Tools4ever.com/Management/Sales	

Filter	Optional value. The Filter parameter specifies a set of attributes that restricts the contacts that are returned by the query.	Title -like '*Manager'	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller.	EXCHSERVER.tools4ever.com	

Result size	Optional value. The ResultSize parameter sets the maximum number of results to return. If all contacts should be returned, use "unlimited" for the value of this parameter. The default value is 1000.	1000	
Identity	The Identity parameter identifies the contacts. If this parameter is specified, only the specified contact is returned.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\JSmith, JSmith jsmith@tools4ever.com or the GUID	
Sort by	Optional value. The SortBy parameter sorts by a single attribute in ascending order.	Name	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
ContactTableSimple	The resulting table with columns for Name, DisplayName and DistinguishedName for each contact.		
ContactTableRegular	The resulting table with columns for Name, DisplayName, DistinguishedName, RecipientType, and Identity for each contact.	%ContactTableRegular%	

ContactTableAdvanced	The resulting table with many columns for each contact.		The columns available are: AssistantName, City, Company, CountryOrRegion, Department, DirectReports, DisplayName, DistinguishedName, ExchangeVersion, Fax, FirstName, Guid, HomePhone, Identity, Initials, IsValid, LastName, Manager, MobilePhone, Name, Notes, ObjectCategory, ObjectClass, Office, OriginatingServer, OrganizationalUnit, OtherFax, OtherHomePhone, OtherTelephone, Pager, Phone, PhoneticDisplayName, PostalCode PostOfficeBox, RecipientType, RecipientTypeDetails, TelephoneAssistant, SimpleDisplayName, StateOrProvince, StreetAddress, Title, UMDialPlan, UMDtmfMap, WebPage, WhenChanged, WhenCreated.
----------------------	---	--	--

*Distribution group*

Script Action: Create distribution group (Exchange 2007)

**Function**

Create a new distribution group of the following types: 'Mail-enabled universal security group', 'Universal distribution group'. To run the New-DistributionGroup cmdlet, the account you use must be delegated the following: 'Exchange Recipient Administrator role', 'Account Operator role for the applicable Active Directory containers'..

**Properties**

Property Name	Description	Remarks
Name	This parameter specifies the name for the new distribution group. The value that is specified in the Name parameter is also used for DisplayName if the DisplayName parameter is not specified. The Name value can't exceed 64 characters.	
SAM Account Name	This parameter specifies the name for clients of the object that are running older operating systems. The SamAccountName parameter is displayed in Active Directory and the Exchange Management Console in the Group name (pre-Windows 2000) field.	
Type	Optional Value. This parameter specifies the group type that will be created in Active Directory. The group's scope is always Universal. Valid values are 'Distribution' or 'Security'. 'Distribution' is the default value.	

Alias	Optional value. This parameter can be used to specify the alias of the distribution group. The Alias parameter is then used to generate the primary SMTP e-mail address of the object. The value of Alias can't contain spaces. If Alias is not specified, the value of SAMAccountName is used to generate the primary SMTP e-mail address, with any spaces converted to underscores.	
Display name	Optional value. This parameter can be used to specify the name of the distribution group in the Exchange Management Console and in the Exchange GAL. If the DisplayName parameter is not specified, the value of the Name parameter is used for DisplayName.	
Managed by	Optional value. This parameter can be used to specify the name of the mailbox user, mail-enabled group, or mail-enabled contact that appears in the Managed by tab of the Active Directory object. You can use any of the following values for this parameter: Distinguished name (DN), Canonical name, GUID, Name, Display name, Legacy Exchange DN and Primary SMTP e-mail address.	
OrganizationalUnit	Optional value. This parameter specifies where to create the distribution group in Active Directory by using canonical name syntax.	
Domain controller	Optional value. To specify the fully qualified domain name (FQDN) of the domain controller that writes this configuration change to Active Directory, include the DomainController parameter.	

GUID	Optional value. This parameter returns the GUID of the new mail distribution group.	Output parameter
Legacy Exchange DN	Optional value. This parameter returns the Legacy Exchange DN of the new mail distribution group.	Output parameter

Script Action: Enable distribution group (Exchange 2007)

#### Function

This action is used to mail enable an existing universal group. To run this action, the account you use must be delegated the "Exchange Recipient Administrator" role.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter specifies the identity of the distribution group in one of the following forms:  * GUID * DN * Domain\Account Name	'CN=jsmith,OU=sales,DC=tools4ever,DC=com', 'tools4ever.com\jsmith' or the GUID.	
Alias	The Alias parameter specifies the e-mail alias of the group.		



DisplayName	The DisplayName parameter specifies the display name of the distribution group. A display name is typically the same as the Domain\Account Name.		
PrimarySmtpAddress	Use this parameter to specify the primary SMTP address for the distribution group. By default, the primary SMTP address is generated based on the default e-mail address policy. If you specify a primary SMTP address by using this parameter, the cmdlet will set the EmailAddressPolicy Enabled attribute of the distribution group to \$false, and the e-mail addresses of this distribution group will not be automatically updated based on e-mail address policies.		

DomainController	To specify the fully qualified domain name (FQDN) of the domain controller that writes this configuration change to Active Directory, include the DomainController parameter in the command.		
------------------	--	--	--

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
GUID	The GUID of the existing mail distribution group.		
Legacy Exchange DN	This parameter returns the Legacy Exchange DN of the existing mail distribution group.		

Script Action: Edit distribution group (Exchange 2007)

**Function**

Use this action to modify the settings of an existing distribution group. Distribution groups are used to consolidate groups of recipients into a single point of contact for e-mail messages. Distribution groups cannot be used to assign permissions to network resources in the Active Directory directory service. Use this action to overwrite existing settings or to add new settings for an existing distribution group. To run this action, the account you use must be delegated the following: 'Exchange Recipient Administrator role'.

All properties of the action are described in the action property dialogs.

Script Action: Disable distribution group (Exchange 2007)

**Function**

This action is used to remove mail capabilities from an existing mail-enabled group. To run this action, the account you use must be delegated the "Exchange Recipient Administrator" role.

All properties of the action are described in the action property dialogs.

Script Action: Remove distribution group (Exchange 2007)

**Function**

This action deletes an existing distribution group from the Active Directory directory service. To run the action successfully, the account used must be delegated the following:

- Exchange Recipient Administrator role
- Account Operator role for the applicable Active Directory containers

All properties of the action are described in the action property dialogs.

Script Action: List distribution groups (Exchange 2007)

#### Function

Retrieve information about distribution groups. The action returns a table with rows for each distribution group. To run the action successfully, the account you use must be delegated one of the following: 'Exchange View-Only Administrator role'.

All properties of the action are described in the action property dialogs.

#### *Mailbox*

Script Action: List mailbox statistics (Exchange 2007)

#### Function

Retrieve information about mailboxes. The action returns a table containing information such as the size of a mailbox, the number of messages and the last time it was accessed.

#### Properties

Property Name	Description	Typical setting	Remarks
Database	Optional value. The Database parameter specifies the name of the mailbox database. When a value is specified for the Database parameter, the Exchange Management Shell returns statistics for all the mailboxes on the database specified. Do not use this parameter in conjunction with the 'Server' parameter.	'Mailbox database', 'SERVERNAME\Mailbox Database' or 'SERVERNAME\Storage Group Name\Mailbox Database'	

Server	<p>Optional value. The Server parameter specifies the server from which you want to obtain mailbox statistics. Use for example: the Fully qualified domain name or the NetBIOS name. When a value for the Server parameter is specified, the action returns statistics for all the mailboxes on all databases on the specified server. Otherwise, the action returns logon statistics for the local server. Do not use this parameter in conjunction with the 'Database' parameter.</p>	<p>'servername.tools4ever.com' or 'SERVERNAME'</p>	
--------	---	--	--

Disconnect d mailboxes only	Optional value. With the 'Disconnect d mailboxes only' parameter the resulting table will contain only disconnecte d mailboxes.	'Yes' or 'No'	
Above size limit only	Optiona value. With the 'Above size limit only' parameter the resulting table will contain only the mailboxes with a mailbox size above the specified value. Qualify the value with one of the following: B (bytes), KB (kilobytes), MB (megabytes) or GB (gigabytes).	20MB	

Above item number only	Optional value. With the 'Above item number only' parameter the resulting table will contain only the mailboxes with a number of items above the specified value.	100	
Identity	The Identity parameter identifies the mailbox. If this parameter is specified, only the mailbox that is associated with the specified user, is returned.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\JSmith, JSmith jsmith@tools4ever.com or the GUID	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.



Property	Description	Default variable name	Remarks
Mailbox	The resulting table with columns for DisplayName, Identity, MailboxGUID, DatabaseName, StorageGroupName, ServerName, LegacyDN, ItemCount, TotalItemSize, DisconnectDate and StorageLimitStatus for each mailbox.		

*Exchange server*

Script Action: List mailbox databases (Exchange 2007)

**Function**

Retrieve one or more mailbox database objects from a storage group, server or organization. The action returns a table with rows for each mailbox database.

**Properties**

Property Name	Description	Typical setting	Remarks
Server	Optional value. The Server parameter specifies the name of the server from which to retrieve mailbox database information. If this parameter is specified, the command will retrieve information about all of the mailbox databases on that server. Use the Fully qualified domain name or the NetBIOS name.	'servername.tools4ever.com', 'SERVERNAME'	

Storage group	Optional value. The 'Storage group' parameter specifies the name of the storage group from which to retrieve mailbox database information. If this parameter is specified, the command will retrieve information about all of the mailbox databases on the storage group.	'EXCHSERVER\First Storage Group', 'First Storage Group'	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller you want to use.	EXCHSERVER.tools4ever.com	

Identity	Optional value. The Identity parameter identifies the mailbox database.	'EXCHSERVER\First Storage Group\Mailbox Database', 'EXCHSERVER\Mailbox Database', 'First Storage Group\Mailbox Database'	
----------	---	--	--

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
DatabaseTable	The resulting table with columns for Name, StorageGroupName, Server, DistinguishedName, Identity and Description for each mailbox database.	%DatabaseTable%	

DatabaseTableAdvanced	The resulting table with many columns for each mailbox database.		The columns available are: Name, StorageGroupName, Server, DistinguishedName, Identity, Description, Organization, ExchangeLegacyDN, EdbFilePath, PublicFolderDatabase, AdministrativeGroup, IssueWarningQuota, ProhibitSendReceiveQuota, ProhibitSendQuota, WhenChanged, WhenCreated, ExchangeVersion.
-----------------------	--	--	---

### *Out-Of-Office*

Script Action: Get Out-Of-Office info (Exchange 2007)

#### **Function**

Retrieve the Out Of Office settings of a mailbox on Exchange 2007. The action returns all properties of the Out Of Office settings dialog in Outlook 2007.

This action can only work properly with Exchange Web Services. Learn how to use this action by reading topic *Using the Exchange Web Services with UMRA* on page 7

#### **Properties**

Property Name	Description	Typical setting	Remarks
Session ID	The Powershell session id. You can create a powershell session ID by adding the 'Setup Powershell Agent server session' to your script. Do not forget to release the Powershell session with the 'Release Powershell Agent server session' action	%PowershellAgentSessionId%	In the Action tree, navigate to Powershell - Agent service session - Setup Powershell Agent server session.
Email address	The email address of the mailbox on the Exchange server	jsmith@tools4ever.com	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
Out Of Office state	The Out Of Office state of the specified mailbox. Possible values are: Disabled, Enabled, Scheduled		
Out Of Office internal reply	The Out Of Office internal reply text of the specified mailbox. Note that this text is created with HTML to keep its formatting.		
Out Of Office scheduled start date	The Out Of Office scheduled start date of the specified mailbox.		
Out Of Office scheduled end date	The Out Of Office scheduled end date of the specified mailbox.		
Out Of Office external reply state	The Out Of Office external reply state of the specified mailbox. Possible values are: None, Known, All.		When 'None' is specified, only the contacts within your organization are receiving the message. If 'Known' is specified, the external text is only send to the users which are in your 'Contacts' folder in Outlook.

Out Of Office external reply text	The Out Of Office external reply text of the specified mailbox. Note that this text is created with HTML to keep its formatting.		
-----------------------------------	--	--	--

Script Action: Set Out-Of-Office info (Exchange 2007)

#### Function

Set the Out Of Office settings of a mailbox on Exchange 2007. The action sets all properties of the Out Of Office settings dialog in Outlook 2007.

This action can only work properly with Exchange Web Services. Learn how to use this action by reading topic *Using the Exchange Web Services with UMRA* on page 7

#### Properties



Property Name	Description	Typical setting	Remarks
Session ID	The Powershell session id. You can create a powershell session ID by adding the 'Setup Powershell Agent server session' to your script. Do not forget to release the Powershell session with the 'Release Powershell Agent server session' action	%PowershellAgentSessionId%	In the Action tree, navigate to Powershell - Agent service session - Setup Powershell Agent server session.
Email address	The email address of the mailbox on the Exchange server	jsmith@tools4ever.com	
Out Of Office state	The Out Of Office state of the specified mailbox. Possible values are: Disabled, Enabled, Scheduled		

Out Of Office internal reply	The Out Of Office internal reply text of the specified mailbox. Note that this text is created with HTML to keep its formatting.		Outlook will accept text that has no HTML format, previous formatting will be erased though!
Out Of Office scheduled start date	The Out Of Office scheduled start date of the specified mailbox.		Note that the end date cannot be before the scheduled start date. No error will occur, but the End Date, Start Date and Out Of Office State are not set!
Out Of Office scheduled end date	The Out Of Office scheduled end date of the specified mailbox.		Note that the end date cannot be before the scheduled start date. No error will occur, but the End Date, Start Date and Out Of Office State are not set!
Out Of Office external reply state	The Out Of Office external reply state of the specified mailbox. Possible values are: None, Known, All.		

Out Of Office external reply text	The Out Of Office external reply text of the specified mailbox. Note that this text is created with HTML to keep its formatting.		Outlook will accept text that has no HTML format, previous formatting will be erased.
-----------------------------------	--	--	---

### 4.3.2. File System

#### Script Action: Create Directory

##### Function

Creates a directory on a (NTFS) file system. For the directory, you can setup the permissions as well. Additionally, you can create a share for the directory.

##### Deployment

This action is typically used in a script that is intended to create new users in Active Directory or NT4 domains, after creation of the actual user account with *Script Action: Create User (AD)* on page 3 or *Script Action: Create User (no AD)* on page 68 . This action is then used to create for example the home directory and share for that user in the file system. It can however also be used in any other context.

##### Properties

Property Name	Description	Typical setting	Remarks
Computer	The computer name on which the directory is created	%HomeServer%	See the <b>Remarks</b> section below.

Parent path	The relative path to the parent directory of the directory to be created	users	See the <b>Remarks</b> section below. The path has the form <share name>\<subdir1>\... Example: users\students\2004
Directory name	The name of the directory to be created	%UserName%	See the <b>Remarks</b> section below. The directory will be created as a sub-directory of the specified parent path.
Always create unique directory	Add a number to the directory name before creating the directory, if a directory with the original name does already exist.	Yes	
Security	Specifies the (NTFS) access rights on the Directory	Set by special dialog	Specifies the access rights for different users on the directory. It is possible to use variables to construct the names.  It is also possible to use a variable that contains the SID of a user instead of a user name. When creating a user with the script action Create User, the SID of the user is exported to the variable %UserSid% by default. This variable can be used inside the dialog to refer to the just created user.
Share the directory	Specifies if the directory must be shared.	No	

Share name	The name by which the directory is shared.	%UserName% or %UserName%\$	In order to create a hidden share, specify a \$ as the last char of the name
Share permissions	The permissions of the share (!) of the new directory. If this property is not specified, the default settings apply.		If the permissions of the share are not specified, the share permissions are set to full control for everyone.
Share user limit	Specifies the number of users who can connect to the shared folder at one time. If this property is not specified, the number is set to unlimited.		If not specified, an unlimited number of user connections is accepted.

#### Remarks

A directory is always created in a **parent** directory. The directory can be created on a remote or the local computer. **The parent directory must be accessible in order to successfully create the directory.** Further, the user running the application must have sufficient access rights for the parent directory to create the directory and setup the share and permissions. The parent directory has the following format (specified using property names):

#### Computer\Parent path

The field **Computer** specifies the name of the computer in NETBIOS or DNS-style. The **Parent path** specifies the name of a share and eventually a directory on the **Computer**. The table below shows some examples. In this table the following columns are shown:

**Computer:** property of the action;

**Parent path:** property of the action;

**Local path on specified computer:** The logical drive path of the resulting total path of the parent directory. This path is relative to the specified computer.

**Resulting total path of parent directory:** The target directory is created in this directory. Using this specification, the parent path can be access from a remote computer.

**Comments:** Description of this example entry.

Computer name	Parent path	Local path on specified computer	Resulting total path of parent directory	Comments
SERVER_A	Users	G:\Users	\\SERVER_A\Users	The directory <b>G:\Users</b> is shared as <b>Users</b> on the computer.
SERVER_A	Users\Sales	G:\Users\Sales	\\SERVER_A\Users\Sales	The directory <b>G:\Users</b> is shared as <b>Users</b> on the computer. The directory <b>Sales</b> is a subdirectory of this directory.
SERVER_A	Sales	G:\Users\Sales	\\SERVER_A\Sales	The directory <b>G:\Users\Sales</b> is shared as <b>Sales</b> on the computer.

SERVER_A	G\$\Users	G:\Users	\\SERVER_A\G\$\Users	The local drive <b>G:\</b> on the computer is shared as <b>G\$</b> (administrative share). The directory <b>Users</b> is a subdirectory of logical drive <b>G:\</b> .
----------	-----------	----------	----------------------	---

### Script Action: Get file/directory info

#### Function

Using the **Get file / directory info** script action, specific info regarding a file or directory can be retrieved (e.g. to check whether a file or directory exists or not) and used in subsequent script actions.

#### Properties

Property Name	Description	Typical setting	Remarks
Target file/directory	The name of the target file or directory.	Can be specified using the full UNC path (recommended) or local path (local to the UMRA service or UMRA console application)	Output only property
File/dir exists flag	When specified, the output variable value is set to YES if the file or directory exists.		Output only property

Is directory flag	When specified, the output variable value is set to YES if the target path identifies a directory.		Output only property
Attributes	When specified, the output variable is set equal to the attribute mask of the file or directory.		Output only property
Creation time	When specified, the output variable is set to the creation time of the file or directory.		Output only property
Last access time	When specified, the output variable is set to the last access time of the file or directory.		Output only property
Last write time	When specified, the output variable is set to the last write time of the file or directory.		Output only property
Size	When specified, the output variable is set to the size of the file in bytes		Output only property. Meaningless value in case of directories.



Error if not found	Set this property to "Yes" if this action should produce an error in case the file or directory does not exist.	Default value is "No"	Output only property
--------------------	---	-----------------------	----------------------

### Script Action: Copy directory

#### Function

Copies the contents of one directory to another directory. The source and destination directory can reside on different computers. A number of options are available: create the destination directory, setup permissions, copy permissions etc.

#### Deployment

This action is typically used in a script that is intended to manage existing user accounts and move for instance home directories. By combining the action **Copy directory** and *Delete directory* on page 357 the **Move directory** action can be implemented. Besides copying the files and directories, the security permissions can be setup for the destination directories and files. For the permissions, three options are available:

1. **Copy security settings from source directory:** All permissions settings are copied for each individual file and directory. To select, set the property **Copy security option** to **Yes** and property **Setup security option** to **No**.
2. **Setup security settings for destination directories and files:** Initialize the security settings for the destination files and directories. The security settings of the source directories and files are not used. Instead, you can specify the new security settings for the destination files and directories. To select, set the property **Copy security option** to **No** and property **Setup security option** to **Yes** and specify the security settings with property **Security**.

3. **No configuration:** The copy operation is executed but not security settings are explicitly setup. The security settings of the destination directory and files are determined by the security settings of the destination parent directory and inheritance rules. To select, set the property **Copy security option** to **No** and property **Setup security option** to **No**. This is the default option.

#### Properties

Property Name	Description	Typical setting	Remarks
Source directory	<p>The name of source directory. The source directory can be specified in two ways: For local directories: <b>&lt;logical drive&gt;\\&lt;directory&gt;\\&lt;directory&gt;</b> etc.  Example: 'C:\\UserData\\Marketing'.</p> <p>For remote and local directories:  <b>\\\\&lt;computer&gt;\\ &lt;share&gt;\\ &lt;directory&gt;\\ &lt;directory&gt;</b>.  Example: '\\\\SERVER_A\\Users\\Data'. The source directory must exist.</p>		

Destination directory	<p>The name of destination directory. The destination directory can be specified in two ways: For local directories: <b>&lt;logical drive&gt;\\&lt;directory&gt;\\&lt;directory&gt;</b> etc. Example: '<b>C:\\UserData\\Marketing</b>'. For remote and local directories: <b>\\\\&lt;computer&gt;\\&lt;share&gt;\\&lt;directory&gt;\\&lt;directory&gt;</b>. Example: '<b>\\\\SERVER_A\\Users\\Data</b>'. If the destination directory does not exist, it can be created.</p>		<p>If the destination directory does not exist, it can be created by setting property Create destination directory to Yes. This will create the full path if necessary.</p>
Create destination directory	<p>A flag indicating that the destination directory must be created if it does not exist. Default value: '<b>Yes</b>'.</p>	Yes	<p>If not specified, the default value Yes is applied.</p>
Copy subdirectories	<p>Specify '<b>Yes</b>' to copy the complete directory tree, including subdirectories and files, and subdirectories of subdirectories.</p>	Yes	<p>If not specified, the default value Yes is applied.</p>
Copy directories, no files	<p>Specify '<b>Yes</b>' to copy directories only, no files. Default value: '<b>No</b>'. If you specify '<b>Yes</b>', no files are copied, only the directory tree is copied to the destination directory.</p>	No	<p>If not specified, the default value No is applied.</p>

Use backup and restore privileges	A flag indicating that backup and restore privileges must be used to copy the directory. This property is required in case the logged on user has no access rights to the directories and files that must be copied. The logged on user must have the corresponding access rights configured on the target computer to use these privileges successfully. Default value: 'Yes'.	Yes	If not specified, the default value Yes is applied. The access rights are configured using policies. Depending on the environment, Domain security, Domain Controller Security or Local Security policies apply. The backup and restore privileges are configured by settings the Backup files and directories and Restore files and directories policies of the User Rights Assignment for the logged on
-----------------------------------	---	-----	---

Continue on error	A flag indicating that the copy directory action must continue if an error occurs when copying a file or directory. Default value: ' <b>Yes</b> '.	Yes	If set to Yes the copy action continues , but an error will be reported and is returned by the action.
Overwrite existing files	A flag indicating that existing destination files must be overwritten if they already exist. If you specify ' <b>No</b> ' instead, an error is generated and the file is not overwritten. Default value: ' <b>Yes</b> '.	Yes	
Copy security option	Copy security settings from the source directory and files to the destination directory and files. The security settings include the access rights, owner and auditing settings. If the security is not explicitly specified, the security settings of the destination parent directory determine the new security settings.		See Deployment section.
Setup security option	Setup the security settings for the target directory and files. The security settings include the access rights, owner and auditing settings. The security settings are specified with the property <b>Security settings</b> . If the security is not explicitly specified, the security settings of the destination parent directory determine the new security settings.		See Deployment section.

Security	The new security settings for the target directory and files. If you want to use this option, you must set the value of property ' <b>Setup security option</b> ' to 'Yes'.	This property is only used when the value of property Setup security option is set to Yes. For more information, see <i>Security - Overview</i> on page 754.
----------	---	--

**See also:**

*Script Action: Create Directory* on page 341

*Script Action: Delete directory* on page 357

**Script Action: Rename file or directory****Function**

Renames the name of a file or a directory (e.g. a home directory for a user)

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts. With this action you can rename a home directory for a user or to move user files to a different location.

**Properties**

<b>Property Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
Source file / directory	The full path of the original file or directory		
Destination file / directory	The full path of the destination file or directory		Both for files and directories, the target directory must exist.
Allow the move to different volume flag	Allows moving the file to a different volume. The default value is "Yes"	Yes	This option can only be used for files, not for directories.
Delay rename until reboot flag	Specifies that the file should not be moved until the operating system has been restarted. The default value is "No".	No	
Replace existing file flag	Replaces the destination file if it already exists. The default value is "Yes".	Yes	This option can only be used for files, not for directories.
Flush before return flag	The script action remains active until the move has been completed and the data written to disk. The default value is set to "No."		

## Script Action: Setup Security

### Function

This script action sets up the security access rights for a directory, file or directory tree using the Access Control List.

### Deployment

This function is used in a network environment to mass apply security settings or to apply security settings as part of a delegation project. Setting the security using the **Setup Security** script action in UMRA is essentially the same as setting the security using Windows 2000/2003 access control (ACL), with the one major difference that in UMRA you can use variables for the target directories and files. This allows you for instance to set the security settings for a whole range of user directories in one sweep.

### Properties

Property Name	Description	Typical setting	Remarks
Target directory or file	The name of the source directory or file. The source directory can be specified in two ways.  For local directories: <logical drive>\<Directory>\<Directory> etc.  For remote and local directories: \\<computer>\<share>\<directory>\<directory>.  <b>Example: "\\SERVER_A\Users\Data".</b>		The source directory must exist. The name of the file or directory should not include any wildcard characters.
Security	The new security settings for the target directory and / or files. Note that the original security settings will be overwritten.		



Propagation mode	Specify "Yes" to set up the security settings for a typical user account home directory. The ACE settings of the specified target directory are explicitly set to the specified value and then propagated to all child directories and subdirectories. All children will inherit the ACE settings of the specified target directory.		Mode to choose when running Windows 2000 or higher versions.
Recursive mode	Specify "Yes" to set up the security settings for the whole directory tree, including the specified directory, subdirectories and files, subdirectories of subdirectories, etc. For all directories and files of the directory tree, the security settings are set identically. The main difference with the propagation mode is that all children of the specified directory will have explicit ACE settings.		Note that you cannot have both "Protected propagation mode" and "Recursive mode" set to "Yes". This option is suitable for Windows NT environments.
Use backup and restore privileges	A flag indicating that backup and restore privileges must be used to set up the security settings. This property is required in case the logged on user has no access rights to the directories and files that must be copied. The logged on user must have the corresponding access rights configured on the target computer.	No	Only certain users have restore and backup privileges. This option will only work properly if these privileges have been explicitly set for the currently logged-in account.

Continue on error	A flag indicating that the action must continue with the next file or directory if an error occurs when editing the permissions of a file or directory.	Yes	
-------------------	---	-----	--

### Script Action: Delete file(s)

#### Function

Deletes the specified file or files. The file name specification can include wildcards (\* or ?).

#### Deployment

This script action is typically used as part of a cleanup action (e.g. deleting temporary files or deleting files from a user profile directory).

#### Properties

Property Name	Description	Typical setting	Remarks
File path	The full path of the file(s). The path should start with a logical drive (e.g. G:\) or share name \\<Computer>\<Share>. The path name may contain the wildcard characters * and ?.		
Include subdirectories (recursive)	If this property is set to "Yes", all files in the child subdirectories of <File path> will be deleted as well.	No	
Use backup and restore privileges	A flag indicating that backup and restore privileges must be used to delete the files. This property is required in case the logged on user has no access rights for the directories and files that must be deleted. If this option is set to "Yes", the logged on account must have rights for "Restore files and directories" and "Back up files and directories" in the local policy.	No	Only needs to be set to "Yes" if currently logged on user does not have sufficient rights to delete the specified files.

Ignore error		No	
--------------	--	----	--

### Backup and restore privileges

In Windows NT/2000/XP you must have Backup and Restore privileges to delete the required files. Add backup and restore privileges. If you use a local system account to delete files, this account should have the necessary privileges for deleting files or this script action will fail.

To check an account's privilege , you can:

1. Click " **Start->Control Panel->Administrative Tools->Domain Security Policy**".
2. Expand the **Local Policies** folder. Select **User Rights Assignment** .
3. The user account should be specified under the policies "**Restore files and directories**" and "**Back up files and directories**".

### See also:

*Script Action: Delete directory* on page 357

### Script Action: Delete directory

#### Function

Deletes the directory tree, including all files and subdirectories. Optionally, you can delete the specified directory itself. The directory tree is specified by a single directory name. The name must have the syntax: \\\\COMPUTER\\Share\\Directory\_To\_Delete or DRIVE:\\Directory\\Directory\_To\_Delete (local drive). If the directory to delete corresponds with a remote share (\\\\COMPUTER\\Shared\_Dir\_To\_Delete), you can use the administrative share (\\\\COMPUTER\\DRIVE\$) to delete the directory tree.");

#### Deployment

This action is typically used in a script that is intended to remove user accounts and all of the associated resources. More generally, the action can be used to delete one or more directory trees. To access the directory that must be deleted, a share is used. In case the specified directory must be deleted as well, and the directory is specified as

\\SERVERNAME\ShareName, the specified share cannot be used to delete the directory. In this case, another share must be used to delete the directory. By default, the administrative share (\\SERVERNAME\C\$, \\SERVERNAME\D\$) is then used to delete the directory. If this share cannot be used, an error occurs.

### Properties

Property Name	Description	Typical setting	
Directory name	The name of the directory tree to delete. All files, directories and subdirectories will be deleted from the specified directory. Optionally, you can delete the specified directory itself. The directory name must have the syntax: \\\\\\COMPUTER\\Share\\Directory_To_Delete or DRIVE:\\Directory\\Directory_To_Delete. If the directory to delete corresponds with a remote share (\\\\\\COMPUTER\\Shared_Dir_To_Delete), you can use the administrative share (\\\\\\COMPUTER\\DRIVE\$) to delete the directory tree.		
Delete directory option	A flag indicating if the specified directory itself (property: 'Directory name') must be deleted.	No	If not specified, the default value <b>No</b> is applied.
Delete read-only files and directories	A flag indicating if read-only files and directories must be removed as well. Default value: TRUE. To delete the read-only files and attributes, the read-only attribute is reset first.	Yes	If not specified, the default value <b>Yes</b> is applied.

Always use administrative share	A flag indicating that the administrative share must be used to delete the directory. The administrative share is by default only used if the specified directory must be deleted and the directory corresponds with a share (Syntax: \\\COMPUTER\\Shared_Dir_To_Delete).	No	If not specified, the default value <b>No</b> is applied.
Never use administrative share	A flag indicating that the administrative share should not be used to delete the directory. The administrative share is by default only used if the specified directory must be deleted and the directory corresponds with a share (Syntax: \\\COMPUTER\\Shared_Dir_To_Delete).	No	If not specified, the default value <b>No</b> is applied.

Use backup and restore privileges	A flag indicating that backup and restore privileges must be used to delete the directory. This property is required in case the logged on user has no access rights to the directories and files that must be deleted. The logged on user must have the corresponding access rights configured on the target computer to use these privileges successfully.	No	If not specified, the default value <b>Yes</b> is applied. The access rights are configured using policies. Depending on the environment, <b>Domain security</b> , <b>Domain Controller Security</b> or <b>Local Security</b> policies apply. The backup and restore privileges are configured by settings the <b>Backup files and directories</b> and <b>Restore files and directories</b> policies of the <b>User Rights Assignment</b> for the logged on user account.
-----------------------------------	--	----	---

Ignore error	A flag indicating that errors must be ignored when a directory tree is deleted.	No	This flag can be used to prevent error messages when a directory for instance does not exist.
--------------	---	----	---

**See also:**

*Script Action: Create Directory* on page 341

*Script Action: Copy directory* on page 347

**Script Action: Create share****Function**

Creates a share on a directory or disk. Using this function you can set the share permission and user limit as well.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts and move user home directories. When you use *Script Action: Copy directory* on page 347 no share is created. When you want to share a directory, the **Create share** action should be applied. A share is typically used to connect to network data that should be available for a group of users.

**Properties**

Property Name	Description	Typical setting	Remarks
Share path	<p>The full path of the directory that is going to be shared. Both remote and local directories can be shared.</p> <p>The share path can be specified in two ways: For local directories: &lt;logical_drive&gt;\&lt;directory&gt;\&lt;directory&gt; etc. Example: 'C:\UserData\Marketing'. For remote and local directories: \\&lt;computer&gt;\&lt;share&gt;\&lt;directory&gt;\&lt;directory&gt;. Example: '\\SERVER_A\Users\Data'. The directory which is going to be shared must exist.</p>	%SharePath %	
Share name	<p>The name given to the share. The name must be unique with respect to other shares on the computer.</p>	%ShareName%	<p>You should always use a name that is easily identified.</p> <p>An user homedirectory for example would be easily identified by the username.</p>



Make share name unique	Makes the share name unique. A share name must always be unique, when the share name is not unique the share will not be created.		A number is added to make the share name unique. The number starts with 1 and will increase till an unique name is found. When this property is not set the share will not be created when the share name already exists.
Share permissions	The permissions of the share (!) of the new directory. If this property is not specified, the default settings apply.		If the permissions of the share are not specified, the share permissions are set to full control for everyone.
User limit	Specifies the number of users who can connect to the shared folder at one time. If this property is not specified, the number is set to unlimited.		If not specified, an unlimited number of user connections is accepted.

**See also:**

*UMRA Basics* on page 3

*Script Action: Delete share* on page 366

*Script Action: Edit share* on page 364

**Script Action: Edit share****Function**

Edits the properties of an existing share. Using this function you can also add comments for a share.

**Deployment**

This action is typically used in a script that is intended to manage existing user accounts and move user home directories.

**Properties**

Property Name	Description	Typical setting	Remarks
Computer	Name of the computer maintaining the share you want to edit		
Share name	The name of the share that must be updated. Note that the name of the share is not necessarily equal to the name of the shared directory.	%ShareName%	This is the name given to the share you want to edit.
Share comment	Here you can add a comment for the shared folder. The comment can only contain text and is an optional field.		

User limit	Specifies the number of users who can connect to the shared folder at one time. If this property is not specified, the existing settings will apply.		If not specified, the existing settings for the share will apply
Share permissions	The new permissions for the share. If this property is not specified, the existing settings apply.		If the permissions of the share are not specified, the share permissions are set to full control for everyone.
Cache parameter	The parameter specifies the caching of the contents of the share available to users who are offline. Possible values:  0 : Cache only specified files and programs (default); 16 : Cache files and programs opened by users; 32 : Cache files and programs opened by users (optimized for performance); 48 : Disable caching.	0,16,32 or 48	If the parameter is not specified, the cache settings are not updated.

**See also:**

*UMRA Basics* on page 3

*Script Action: Create share* on page 361

*Script Action: Delete share* on page 366

## Script Action: Delete share

### Function

Deletes a share from a directory or disk. This action only removes the share of a directory or disk, it does not remove the directory or disk. Use *Script Action: Delete directory* on page 357 to delete a directory.

### Deployment

This action is typically used in a script that is intended to remove a users account in Active Directory or NT4 domains, after removing of the actual user account with *Script Action: Delete user (AD)* on page 55 or *Script Action: Delete user (no AD)* on page 86. This action is then used to remove for example the share on the home directory. It can however also be used in any other context.

### Properties

Property Name	Description	Remarks
Computer	The name of the computer that maintains the share.	The name of the computer can be specified in NETBIOS or DNS-style (e.g. SERVER_A, server_a.my_domain.com)
Share name	The name by which the shared directory is identified.	In order to remove a hidden share, specify a \$ as the last char of the name. The share name is not necessarily the name of the shared directory.
Ignore error	When this flag is set to 'Yes' and the specified share can not be deleted, no error will be generated.	This option can be used to prevent the script from stopping when an error is generated.

### See also:

*UMRA Basics* on page 3

*Script Action: Create share on page 361*

*Script Action: Edit share on page 364*

### **Script Action: List files and/or directories**

#### **Function**

This script action creates a table with files and / or directories. The result is stored in an output variable.

#### **Deployment**

This script action will typically be used in a delegation project with multiple forms to obtain file and directory info and display the result in a form table. Project A will contain this script action. In Project B, you need to define Project A as an initial project. Before the form of project B is generated, the script of project A is executed and the result is stored in a user defined variable. This variable can then be used in the form fields in project B (e.g. in a generic table Variable).

#### **Setting up the script action**

<b>Name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
Path	Path to the directory of which you want to collect the underlying files and/or directories.	User defined	If you use the browse button, select a file within the target directory and clear the file name.
Include files	Set this option to "No" if you do not want to include files.	Yes	
Include directories	Set this option to "No" if you do not want to include directories.	Yes	

Include subdirs	Set this option to "No" if you do not want to include subdirectories.	Yes	
Use backup privileges	A flag indicating that backup and restore privileges must be used to set up the security settings. This property is required in case the logged on user has no access rights to the directories and files that must be copied. The logged on user must have the corresponding access rights configured on the target computer.	No	
Output variable		User defined	

For each returned user object in the variable **%FileDirs%** (or any other name you may have given to this output variable), the following columns are included:

Column	Description
Full path	Full path to the file or directory
Size (bytes)	Size of the file in bytes
Size (MB)	Size of the file in MB
Directory	Specifies if the object is a directory (Yes/No)
Hidden	Specifies if the file is hidden
Read only	Specifies if the files is Read only
Attributes	Specifies the attribute mask of the file or directory.
Creation time	Creation time of the file or directory.
Access time	Last access time of the file or directory.

Write time	Last write time of the file or directory.
Name	Name of the file or directory

If you want to use the content of the variable in a generic table, you need to set up a generic table of the Variable type. In the setup procedure, you can select the column template **Files and or directories list** which includes the above mentioned columns.

**See also:**

*Script Action: Get file/directory info on page 345*

#### **4.3.3. Other actions**

##### **Script Action: Execute Command Line**

###### **Function**

Executes a Windows command line on the local computer that runs User Management. The command line can contain any number of arguments, including variables.

###### **Deployment**

This action is typically used in a script that is intended to create new users in Active Directory or NT4 domains, usually as one of the last script commands issued. This action is then for instance used to copy some standard files in the user's home directory, or perform some other site specific batch commands related to the just created account.

**Properties**

Property Name	Description	Typical setting	Remarks
Command Line	The command line that must be executed		The command line starts with a name of a file that can be executed (.exe .bat etc), followed by options as required by the specific command. It may be required to specify the complete path the the file to be executed.
Wait until terminated	Specifies whether or not User Management waits for the command to finish before it continue with the next action of the script.	Yes	When set to <b>Yes</b> , the execution of the script is suspended until the command has finished, either successfully or unsuccessfully
Show command window	Specifies whether a command window must show.	No	
Output variable	Name of the variable containing the output generated by the executed command line.	User defined	



Remove carriage-return-line-feed	An option (Yes/No) to remove ending carriage-return-line-feed characters from the value of the Output variable		Often, the output of a command line ends with (one or more) carriage return line feed characters. These characters are not used by subsequent actions that use the variable. By setting this property to 'Yes', the characters will be removed. Note that only the ending characters are removed. The default value (or when not specified) of this property is 'No'.
Output buffer size	Optional: When specified, the application reserves a buffer of the specified size in bytes to store the output data in a variable. By default (not specified), the maximum size is 10240 bytes. If the real output data size exceeds the limit, the data is not stored. Example value: 250000.		
Do not show in log file	Optional: Specify 'Yes' to prevent the command line from being shown in the log file.	not specified (implies 'No')	

**Script Action: Count licensed - domain/OU accounts****Function**

Calculates the number of user accounts in the domains or organizational units for which a licence is configured and compares the actual numbers with numbers allowed for the specific license.

### Deployment

This action is typically used in a script that is executed on a regular basis, for instance by the UMRA scheduler, in order to warn the administrator when the number of accounts in the network approach the values as specified in the license. This way the network administrator can be notified of expected future license violations, and take appropriate actions to prevent this situation.

### Properties

Property Name	Description	Typical setting	Remarks
Available	The number of user accounts that can be created before the count for the license is exceeded.		Output only The number shown is the number for the domain or organizational unit with the least number of available user accounts. The name of this domain or organizational unit is listed in the Domain/OU property
Domain/OU	The domain or organizational unit for which a license is configured, which has the smallest number of available user accounts		Output only
Account License count table	A table containing license usage information for all configured licenses		Output only

**Account License count table**

For each UMRA license as specified in the license configuration of UMRA, a row is added to the table. The table has the following information:

**Domain/OU**

The domain of organizational unit as specified by the license

**Maximum**

The number of user accounts allowed by the license in this domain or organizational unit.

**In Use**

The number of user accounts currently counted in this domain or organizational unit.

**Available**

The number of user accounts that can be created in this domain or organizational unit before the license is exceeded.

**Remarks**

When the a project that contains this action is run in a console mass project, the license situation of the console is collected. In order to collect licence information from a UMRA service, use this action in a project that is executed by the specific server of interest.

**4.3.4. Windows computer services****Script Action: List services status****Function**

With this script action, the name and status of services and drivers installed on a computer are collected and stored in an output variable. Once you have collected these data, you can manage the listed services using the Execute service command script action.

## Deployment

Although this script action can be used in all UMRA modules, the most common usage would be an F&D implementation to collect and manage services. This would require two projects, Project A and Project B. Project A will contain the script action **List services status** to retrieve the services information. In Project B you define the columns you wish to display and the associated actions (Stop service, Start service, etc.)

## Properties

Property Name	Description	Typical setting	Remarks
Computer	The name of the computer where the services / drivers have been installed		
Include services	Select "No" if you do not wish to include the name and status of services in the output table.	Default value is "Yes"	Optional
Include drivers	Select "Yes" if you wish to include the name and status of drivers in the output table.	Default value is "No"	Optional
Include non-stopped	Select "No" if you do not wish to include services and drivers that are NOT in the stopped state	Default value is "Yes"	Optional

Include stopped	Select "No" if you do not wish to include services and drivers that are in the stopped state	Default value is "Yes"	Optional
Include configuration info	Select "Yes" if you wish to include configuration info in the output table.	Default value is "No"	Optional - If "Yes" is selected, the columns "startup type" (text), "startup type" (Code), "binary file" and "logon as" will be added to the output table
Services table	%ServicesTable% is the default name of the output variable containing the list of services	%ServicesTable%	

The output variable (by default %ServicesTable%) may contain the following columns:

Column name (key name)	Description
Computer	Name of the computer where the drivers / services are installed
Internal name	Key name for the service
Name	A Windows service has two names. The long name you see in the Control Panel is the display name of the service. The internal shorter name of the service is called the key name.
Status (text)	Returned as text

Status (code)	Returned as code 1 - Service stopped 2 - Service start pending 3 - Service stop pending 4 - Service running 5 - Service continue pending 6 - Service pause pending 7 - Service error
Process ID	
Svc type (text)	Returned as text
Svc type (code)	Returned as code 1 - Kernel driver 2 - File System Driver 16 - Own Process 32 - Shared Process
Interactive	Yes or No

Please note that a Windows service has two names. The long name as shown in the Control Panel is the display name of the service. The internal, shorter name of the service is called the key name. When you specify the name of a column, the key name must be used.

It is also possible to retrieve the configuration details of each service. If this option is selected, the following columns will be added:

Column name (key name)	Description
Startup type (text)	Returned as text
Startup type (code)	Returned as code 0 - Boot start 1 - System start 2 - Auto start 3 - Demand start 4 - Disabled
Binary file	
Logon as	

**Script Action: Execute service command****Function**

Using the **Execute service** command, you can set **Windows** service settings.

**Deployment**

This action is typically used in combination with the *Script Action: List services status* on page 373 to collect and manage services. In Forms & Delegation, you will typically create two projects: one to collect the services and one to manage these.

**Properties**

Property Name	Description	Typical setting	Remarks
Computer	Name of the computer where the services are running	%ComputerName%	In Forms & Delegation, this variable is passed from the project where the services are collected using the List services script action.
Service name	Name of the service	%ServiceName%	In Forms & Delegation, this is the name of the service that is selected by the end- user in the form. When the user selects no service, the variable is empty.
Start service	Sets the value for Start service to "Yes" . If the service has already started, nothing will happen	%ServiceCommand%	Configure according to the desired service action.

Stop service	Sets the value for Stop service to "Yes" . If the service has already started, nothing will happen	%ServiceCommand%	Configure according to the desired service action.
Restart service	Sets the value for Restart service to "Yes" . If the service is running, it will be stopped first.	%ServiceCommand%	Configure according to the desired service action.
Wait for status completion	If set, the action will not complete until the service has the requested state or if the time-out period has expired.		
Time-out (seconds)	Time-out specified in seconds. Used in conjunction with "Wait for status completion" property.		

**See also:**

*Script Action: Configure service on page 379*



## Script Action: Configure service

### Function

With this script action, services can be configured. This script action should be used in combination with the List services script action which collects a table of services.

### Deployment

This action is typically used for (delegating) management of services.

### Properties

Property Name	Description	Remarks
Computer	Name of the computer where the service is running	
Service name	Name of the service.	Note that the name of the service is not equal to the display name of the service.
Set manual startup type	Set this property to <b>"Yes"</b> to set the startup type of the service to "Manual"	
Set Automatic startup type	Set this property to <b>"Yes"</b> to set the startup type of the service to <b>"Automatic"</b>	
Disable service	Set this property to <b>"Yes"</b> to disable the service	
Log on as Local System account	Set this property to <b>"Yes"</b> to let the service log on to the <b>"Local System account"</b> rather than the user account. Set to <b>"No"</b> to let the service log on to a user account.	
Logon account name	The name of the logon account assigned to the service.	

Logon account password	Password of the logon account assigned to the service.	
------------------------	--	--

**See also:**

*Script Action: List services status on page 373*

*Script Action: Execute service command on page 377*

**4.3.5. Managing printers and printer queues****Script Action: List printer documents****Function**

This action collects the list with printer documents from a specified printer. The data are stored in the output variable %DocumentsTable%.

**Deployment**

With User Management Resource Administrator (UMRA) you can let the helpdesk manage printer queues and print jobs. Individual print jobs can be paused, restarted, resumed and deleted. The printer spooler service itself can be restarted.

Although this script action can be used in all UMRA modules, the most common usage will be an F&D project in which a print job for a particular printer can be selected. The user can then press a button to pause, restart, resume or delete the print job.

**Properties**

Property Name	Description	Typical setting	Remarks
Printer	Name of the printer or print server queue.	Syntax is \\<ComputerName>\<PrinterName>	

DocumentsTable	Table containing the document info for the selected printer or printer queue	%DocumentsTable%	Optional
----------------	--	------------------	----------

The output variable (by default %DocumentsTable%) may contain the following columns:

Column name (key name)	Description
DocumentID	ID number of the document
Document	Name of the document
Status (text)	
Owner	
Pages (total)	Total number of pages in the print job.
Pages printed	Contains the number of pages printed
Submitted	Time when the job was submitted to the print queue
Position in queue	Contains the position of this print job in the print queue.
Priority (text)	Priority of the print job
Priority (code)	Priority of the print job
Printer	Name of the printer
Computer	Computer name
Data type	
Status (code)	

**See also:**

*UMRA tables* on page 9

**Script Action: Execute print job command****Function**

Using the **Execute print job command** script action , you can pause, restart, resume or delete printer jobs. To do this, you need to specify the job ID of the printer job which can be obtained using the script action

*Script Action: List printer documents* on page 380.

**Deployment**

This action is typically used in combination with the **List printer documents** script action to collect the documents in the printer queue.

If you use this script within the Forms & Delegation module, you will typically create two projects: one to collect a list of printer documents and one to manage these.

**Properties**

Property Name	Description	Typical setting	Remarks
Printer	The name of the printer or printer queue.	Syntax: \\ComputerName\PrinterName	
Job id	ID number of the target print job	%jobid%	The job ID number can be obtained using the <i>Script Action: List printer documents</i> on page 380.
Pause print job	Set the value of this property to "Yes" to pause the print job	Default setting is "No"	

Restart print job	Set the value of this property to "Yes" to restart the print job	Default setting is "No"	
Resume print job	Set the value of this property to "Yes" to resume the print job	Default setting is "No"	
Delete print job	Set the value of this property to "Yes" to delete the print job	Default setting is "No"	

#### **4.3.6. LDAP directory services**

##### **Script Action: Setup LDAP session**

###### **Function**

This script action is used to initialize a secure or not secure LDAP session with the LDAP Server. The session parameters are stored in a variable that is used in subsequent UMRA LDAP actions.

**Properties**

Property Name	Description	Typical setting	Remarks
LDAP server	The name of the host running the LDAP server. The name must be specified using the TCP/IP address or DNS name. Optionally, the name can be followed by a colon (:) and port number.	%LdapServer%	
LDAP port	The TCP/IP port number of the LDAP server to which to connect. This property is ignored if the property LDAP server includes a port number. If not specified, the default port is used.	Optional	For non-secure LDAP, the default LDAP port is 389, for secure LDAP (SSL), the default port is 636.

SSL encryption flag	If set to "Yes", the session will use SSL encryption to communicate. In this case, appropriate SSL certificates need to be installed on both the LDAP client and server side. If set to "No", the action will establish a plain TCP connection and use plain text (no encryption).	No	It is strongly recommended to use SSL encryption. To implement this option, SSL certificates need to be installed on both the LDAP Client and Server. The methods how to do this, largely depends on the implementation of the operating system and directory service. For Microsoft Active Directory, Novell eDirectory and Linux OpenLDAP the exact implementations are described in the document Manage LDAP directory services with UMRA which is available in the document library on the Tools4ever website. For other systems, a similar approach must be used.
User name	The user name used to connect to the LDAP server. If this property is not specified, no users are authenticated and you will not be able to execute other LDAP actions.		The format and exact name depends on the directory service.
User password	The password for the user specified in property User name. Note that by default, the password is stored with encryption.	Usually the result of the Generate password script action.	If a value is entered manually, it will be encrypted automatically as soon as you enter OK.

Ldap session	An internal data structure representing the resulting LDAP session. This property is an "output only" property and is generated automatically. This property is used as input for other LDAP script actions.		
--------------	--	--	--

### Script Action: Load LDAP modification data

#### Function

This script action is used to initialize all the attributes and attributes values that are required to update the directory service item. The exact attributes and values used vary for each directory service and are determined by the directory service schema;

#### Deployment

When a directory service is updated to create a new item or update an existing item, the operation is always specified by one or more attributes, the attribute value(s) and the type of attribute value modification: add, delete or replace.

To support this mechanism, the script action Load LDAP modification data is used. All attributes, attribute values and value modification types are specified using this action. The result is stored in a variable that holds all the attribute information. The variable is then used in the action to:

1. Create the item with action *Script Action: Add directory service object (LDAP)* on page 387 or,
2. Update the item with the script action *Script Action: Modify directory service object (LDAP)* on page 388.

The action **Load LDAP modification data** does not communicate with the LDAP Server, that is, no session variable is required.



The **LDAP modification data** window is used to specify the LDAP modification data.



*Example of specifying the LDAP modification data.*

In the example shown above, the data is stored in variable %LdapData%. The data holds the modification values for 5 attributes: **objectClass**, **sn**, **givenName**, **homePhone** and **userPassword**. The names of the attributes are specified using their LDAP names as specified in the schema of the directory service. The values for each attribute can be specified using variables. Each attribute can have one or more values.

### **Script Action: Add directory service object (LDAP)**

#### **Function**

The action is used to add a new item to the directory service. The item is identified by its name that must be unique. All other parameters of the item are specified by its attributes. Before you can use this script action, the following actions must have been executed:

- *Script Action: Setup LDAP session* on page 383: Used to store the session (data) in a variable which is then used as input for other LDAP script actions.
- *Script Action: Load LDAP modification data* on page 386: Used to initialize the attributes and attribute values for the new directory service item.

**Properties**

Property Name	Description	Typical setting
LDAP session	A data structure representing a session with the LDAP server. The property is initialized with action Setup LDAP session and passed to this action using a variable.	The default variable name is %LdapSession%
Object name	The distinguished name of the object to modify. Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com.	
Object data	All attributes and values to add the object. The property must be specified as a variable name. This variable is generated by script action <i>Script Action: Load LDAP modification data</i> on page 386.	

**Script Action: Modify directory service object (LDAP)****Function**

This action is used to update one or more attributes of an existing directory service item. The item is identified by its name (Object name) that is specified as a distinguished name. Before you can use this script action, the following actions must have been executed:

1. *Script Action: Setup LDAP session* on page 383: The session (data) is stored in a variable that is used in this action.
2. *Script Action: Load LDAP modification data* on page 386: Initialize the attributes and attribute values for the new directory service item.

**Properties**

Property Name	Description	Typical setting
LDAP session	A data structure representing a session with the LDAP server. The property is initialized with action Setup LDAP session and passed to this action using a variable.	The default variable name is %LdapSession%
Object name	The distinguished name of the object to modify. Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com.	
Object data	All attributes and values to add the object. The property must be specified as a variable name. This variable is generated by script action <i>Script Action: Load LDAP modification data</i> on page 386.	

**Script Action: Delete directory service object (LDAP)****Function**

This script action is used to delete an existing directory service item. Before you can use this script action, an LDAP session must have been initialized using the *Script Action: Setup LDAP session* on page 383

**Properties**

Property Name	Description	Typical setting
LDAP session	A data structure representing a session with the LDAP server. The property is initialized with <i>Script Action: Setup LDAP session</i> on page 383 and passed to this action using a variable.	The default variable name is %LdapSession%.
Object to delete	The distinguished name of the object to be deleted. Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com. If the item to delete does not exist, an error occurs.	

**Script Action: Rename directory service object (LDAP)****Function**

This action is used to change the distinguished name of an entry in the directory service. The action is available only when using an LDAP session with LDAP version 3.

**Properties**

Property name	Description	Typical setting
LDAP session	A data structure representing a session with the LDAP server.	%LdapSession%
Current name	The distinguished name of the existing entry to be renamed. Example: CN=John Smith, OU=Marketing, DC=tools4ever, DC=com.	

New RDN	The new relative distinguished name of the entry to be renamed. Example: CN=James Smith. The relative distinguished name should not contain the name of the container. If the property is not specified, the directory entry is not renamed but can be moved to another parent container as specified with the property <b>New parent</b> .	
New parent	The new distinguished name of the parent of the entry to be renamed. Example: OU=Marketing, DC=tools4ever, DC=com. This property can be used to move the directory entry to another container. If this property is not specified, the directory entry is not moved to another container.	
Delete old name	A flag indicating if the old relative distinguished name should be deleted. Set to FALSE if the old relative distinguished name should be retained.	Default value is <b>TRUE</b>

**Script Action: Search LDAP**

The **LDAP Search** window is used to specify the LDAP search.

**Session**

The variable representing the LDAP Session that is initialized with action Setup LDAP session.

**Result**

The name of the variable that is used to store the result of the search. The search result is always stored as a table. The variable does not need to exist when the action is executed. If it does exist, the old value is overwritten.

**Base (DN)**

The distinguished name of the directory service tree where the search should start. The search is executed at the specified base, and optionally in the immediate or all subtrees of the directory service.

**Filter**

The specification of the filter to perform the search. The standard search specification according to RFC2254 can be used to execute the search.

**Scope**

Base only	Limits the search to the specified base only. The maximum number of matching directory service items is 1.
One level	The search is performed in all entries of the first level below the base entry, excluding the base entry.
Subtree	The search is performed in the base entry and all levels below the base entry.

**Options**

Time out interval	When enabled, the specified value is the time-out value of the LDAP search and the operation time. If disabled, no time-out value is used.
Size limit	When enabled, the maximum number of matching values is limited to the specified value. When disabled, the maximum number of items is not limited.

**4.3.7. Lotus Notes****Script Action: Get certifier****Function**

Accesses a **Lotus Notes certifier** from its ID file, and creates an internal data structure representing the certifier. Required for any subsequent action that need access to a certifier.

**Deployment**

Typically used before using a script action that registers or renames a person in Lotus Notes. For example the *Script Action: Register person* on page 396 requires access to a certifier in order to be able to certify the new created person.

**Properties**

Property Name	Description	Typical setting	Remarks
Certifier ID file	The path to the Lotus Notes certifier ID file that contains the desired certifier.	C:\Lotus\Domino\data\cert.id.	The File must be accessible in the UMRA module (service or console) that runs the script.

Certifier password	The encrypted password of the id file.		<p>The Id file is protected by a password. Specify here the password required to unlock the Id file.</p> <p>The property configuration window can automatically encrypt the entered password. Only the encrypted value is stored in this action.</p> <p>If the password is specified by means of a variable instead of specified directly, it must be encrypted first. Use the action <i>Script Action: Set encrypted variable</i> on page 546 to create an encrypted variable.</p>
--------------------	--	--	---



Expiration date	The expiration date of new certificates that will be generated when this certifier is used to create certificates.		<p>Input property. Any certificates generated with the resulting certifier object variable will expire at the specified time. If this property is not specified, an expiration date of 2 years from the current time is used.</p> <p>If a variable is used to specify the Expiration date, it must be an UMRA date-time variable, and not a text variable.</p> <p>If the certifier is used to register a person, this date consequently specifies the expiration date of the user account.</p>
Certifier	The resulting certifier variable.	%NotesCertifier%	This variable will contain the resulting certifier object. Use this variable in subsequent actions that require a certifier object as input.

**Script Action: Register person****Function**

Creates and registers an new person in Lotus Notes.

**Deployment**

Typically used as part of a script to create new users and resources in Lotus Notes

**Properties**

Property Name	Description	Typical setting	Remarks
Certifier	Variable containing a object that represents the Lotus Notes certifier that will create the certificate (sign the user id) for the new person.	%NotesCertifier%	Mandatory Use the action <i>Script Action: Get certifier</i> on page 392 to obtain the certifier object before using it in this action.
Basics - Registration server	The name of the Lotus Domino registration server.	servername/domino organization	Mandatory This is the complete name of the server as known within Lotus Notes/Domino environment, not the name of the server as known by the OS hosting the Domino service(s).

Basics - First name	The First name of the user.	%FirstName%	Typically the variable %FirstName% is read from a import file specifying the users to create, or specified in a UMRA form.
Basics - Middle name	The Middle name of the user.	%MiddleName%	Typically the variable %MiddleName% is read from a import file specifying the users to create, or specified in a UMRA form.
Basics - Last name	The Last name of the user.	%LastName%	Typically the variable %LastName% is read from a import file specifying the users to create, or specified in a UMRA form.
Basics - Short name	A short name representing the new user.	Not specified	Optional.  A short name in the format FirstInitialLastName is automatically created as you enter the user's name. For example, JSmith is the short name for John Smith. You can modify this field to overrule this setting.
Basics - Password	A password for the new user ID.	%Password%	Passwords are usually automatically generated in advance with the action <i>Script Action: Generate password</i> on page 565, and exported to a file with action <i>Script Action: Export Variables</i> on page 559 .

Basics - Password Quality Scale	A level for the Password Quality scale.	Not specified	Optional value between 0 (weakest) and 16 (strongest)  Default value is 8  See the Lotus Notes/domino documentation for more information.
Basics - Mail system	The mail system that is used by this new Person.	Lotus Notes	If not specified, or set to "none", no mail system is configured for the new account.  For the exact options see the drop down list in the specific action property in the UMRA console application.
Basics - Create Notes ID	flag if a Lotus Notes ID must be generated for this person.	Yes	Typically this is specified as "YES". if not Specified
Mail - Mail server	The Name of the Mail Server.		The name of the mail server, as known within Lotus Notes.  If not specified the same name as the registration server is used.
Mail - Mail file name	The name of the mail database file of the person.		Optional.  If not specified, the path and file name are set to mail/firstinitial><first7character solastname>.nsf.

Mail - Mail file template	The name of the template used for the mail database.		optional  If not specified the domino server chooses a default template.
Mail - Mail file owner access	The level of access that the mail file owner has to the mail database.	Not specified	Optional.  values: Designer, Editor or Manager.  If not configured, the user has the default "editor with delete documents" access to their database.  This option may for example be set to "Editor", to prevent users from deleting documents from their own mail database.  It cannot be used to explicitly set "Editor with delete documents", since only the most global access settings can be specified here.
Mail - Mail forward address	address to forward the mail to.	Not specified	When specifying, Include the domain names for this person. eg. John Smith@Acme@External.  Note that the mail will not also be send to the original mailbox when a forwarding address is specified.
Mail - Create mail file in background	Specifies whether mail files should be created in the background (Yes/No).	No	Optional. If set to Yes, the mail file will be created in the background by the Lotus Notes adminp process. This will speed up the registration process, but the mail functionality of the created accounts is not immediately available.

Mail - Mail ACL manager	Specifies the account that is allowed to manage access control of the user's mail database.	Not specified	Optional. If not configured the Lotus Notes default settings apply.
Mail - Quota limit (MB)	The size limit (MB) of the user's mail database.		Optional. If not configured, no limit applies.
Mail - Warning threshold	The Size in MB above which Lotus Notes issues a warning message.		
Mail - Create full text index	Specifies if a full text index of the mail database should be created (Yes/No).	No	Optional. If set to Yes, a full text index will be maintained for the mail database.
Address - Internet Address	The internet email address assigned to the new person.		Optional.

ID Info - Store ID in Domino directory	Specifies whether the the userID should be stored in the Domino Directory.	Yes	
ID Info - Store ID in file	Specifies the location where to store the users ID file		Users may need access to (a copy of) this file in order to be able to login.
ID Info - Id file name	The file name including the full path to the file that should be created		In file (default location: <datadirectory>\ids\people\user.id). Click Set ID file to change path.
Id Info - Security type USA	The setting determines the encryption strength(Yes/NO)	Yes	<p>Choose either North American (Yes) or International (No). The security type determines the type of ID file created and affects encryption when sending and receiving mail and encrypting data. North American is the stronger of the two types.</p> <p>This field only is only used when the property "Basics - Create Notes ID" is set to "Yes".</p>

Groups - Groups	The Groups of which the new person should become a member		Separate Multiple groups with a ; character.
Other - Profile name	The name of a R5 user profile to assign	Not specified	This is generally not used in newer versions of Lotus notes, as it cannot be used if you are using policies
Other - location	Departem ental or geographi cal location of the user		
Other - Organiza tional unit	The organizati onal unit of the user		Optional. A word that distinguishes two users who have the same name and are certified by the same certifier ID.
Other - Commen t	A comment about the user, regarding the user's registrati on.		



Other - Local admin name	The name of a user who has Author access to the Domino Directory but who does not have the UserModi fier role. This setting allows the local administr ator to edit Person documen ts.		Optional
Limited client flag		Not defined	Optional.
Desktop client flag			

Internet password flag	Specify "Yes" to store the specified password (additionally) for use as the internet password.		
Enforce short name uniqueness flag	Specify "Yes" to enforce uniqueness of the short name.		
Prompt on duplicate person	Indicates the action to take on duplicate person.		Select one of the following values "Error on a duplicate person (default)" "Skip de person registration" "Update the existing addressbook registration"..
Error if ID file exists	"YES " Specifies that the UMRA script action must fail, and the user must not be created, if the ID file does already exists (Yes/No)	Yes	

Person Document	Specifies the UMRA variable in which should be stored the resulting Person Document created as result of this "Register Person" script action.	%PersonDocument%	Output only. Variable name can be specified on the Out tab only.  The resulting variable can be used in any subsequent script action that requires a person document as input.

**Script Action: Register person (advanced)****Function**

Creates and registers an new Person in Lotus Notes. It allows to specify some advanced settings, for instance to allow creating a roaming person in Lotus Notes.

If none of the new persons to create should be roaming, and you do not need the advanced options you can use *Script Action: Register person* on page 396 instead.

**Deployment**

Typically used as part of a script to create new persons and resources in Lotus Notes

**Properties**

Property Name	Description	Typical setting	Remarks
Certifier	Variable containing a object that represents the Lotus Notes certifier that will create the certificate (sign the user id) for the new person.	%NotesCertifier%	Mandatory  Use the action <i>Script Action: Get certifier</i> on page 392 to obtain the certifier object before using it in this action.
Basics - Registration server	The name of the Lotus Domino registration server.	servername/domino organization	Mandatory  This is the complete name of the server as known within Lotus Notes/Domino environment, not the name of the server as known by the OS hosting the Domino service(s).
Basics - First name	The First name of the user.	%FirstName%	Typically the variable %FirstName% is read from a import file specifying the users to create, or specified in a UMRA form.
Basics - Middle name	The Middle name of the user.	%MiddleName%	Typically the variable %MiddleName% is read from a import file specifying the users to create, or specified in a UMRA form.
Basics - Last name	The Last name of the user.	%LastName%	Typically the variable %LastName% is read from a import file specifying the users to create, or specified in a UMRA form.

Basics - Short name	A short name representing the new user.	Not specified	<p>Optional.</p> <p>A short name in the format FirstInitialLastName is automatically created as you enter the user's name. For example, JSmith is the short name for John Smith. You can modify this field to overrule this setting.</p>
Basics - Password	A password for the new user ID.	%Password%	Passwords are usually automatically generated in advance with the action <i>Script Action: Generate password</i> on page 565, and exported to a file with action <i>Script Action: Export Variables</i> on page 559
Basics - Password Quality Scale	A level for the Password Quality scale.	Not specified	<p>Optional value between 0 (weakest) and 16 (strongest)</p> <p>Default value is 8</p> <p>See the Lotus Notes/domino documentation for more information.</p>

Basics - Password Encryption strength	The encryption key that protects the Notes keys that are stored in the user ID file is derived from the password. The stronger the encryption strength of the password, the stronger the encryption key that protects the Notes keys.	Base strength on RSA key Size	<p>There are three options to choose from:</p> <ul style="list-style-type: none"><li>▪ Base strength on RSA key size - encryption strength is determined by the size of the RSA key stored in the ID file. If the RSA key size is less than 1024 bits, the password encryption strength is 64 bits; if RSA key size is 1024 or greater, the password key size is 128 bits.</li><li>▪ Compatible with all releases (64 bits)</li><li>▪ Compatible with 6.0 and later (128 bits)</li></ul>
Basics - Mail system	The mail system that is used by this new Person.	Lotus Notes	<p>If not specified, or set to "none", no mail system is configured for the new account.</p> <p>For the exact options see the drop down list in the specific action property in the UMRA console application.</p>

Basics - Internet password flag	Specify YES for this option to set an Internet password that is stored in each user's Person document and gives users access to Internet services.		
Basics - Enable roaming	Creates roaming capabilities for this person (Yes/No).	Yes	Optional. If set to Yes, it will create roaming capabilities for this person.
Basics - Create Notes ID	flag if a Lotus Notes ID must be generated for this person.	Yes	Typically this is specified as "YES". if not Specified.
Mail - Mail server	The Name of the Mail Server.		The name of the mail server, as known within Lotus Notes If not specified the same name as the registration server is used.
Mail - Mail file name	The name of the mail database file of the person.		Optional. If not specified, the path and file name are set to mail/firstinitial><first7charactersoflastname>.nsf.

Mail - Mail file templat e	The name of the template used for the mail database.		optional  If not specified the domino server chooses a default template.
Mail - Mail file owner access	The level of access that the mail file owner has to the mail database.	Not specified	Optional.  values: Designer, Editor or Manager.  If not configured, the user has the default "editor with delete docurments" access to their database.  This option may for example be set to "Editor", to prevent users from deleting documents from their own mail database.  It cannot be used to explicitly set "Editor with delete documents", since only the most global access settings can be specified here.
Mail - Mail forward address	address to forward the mail to.	Not specified	When specifying, Include the domain names for this person. eg. John Smith@Acme@External.  Note that the mail will not also be send to the original mailbox when a forwarding address is specified.
Mail - Create mail file in backgro und	Specifies whether mail files should be created in the background (Yes/No).	No	Optional. If set to Yes, the mail file will be created in the background by the Lotus Notes adminp process. This will speed up the registration process, but the mail functionality of the created accounts is not immediately available.



Mail - Mail ACL manager	Specifies the account that is allowed to manage access control of the user's mail database.	Not specified	Optional. If not configured the Lotus Notes default settings apply.
Mail - Quota limit (MB)	The size limit (MB) of the user's mail database.		Optional. If not configured, no limit applies.
Mail - Warning threshold	The Size in MB above which Lotus Notes issues a warning message.		
Mail - Create full text index	Specifies if a full text index of the mail database should be created (Yes/No).	No	Optional. If set to Yes, a full text index will be maintained for the mail database.
Mail - Mail file replicas	Specifies the servers on which the replica(s) of the users's mail files are stored.		Optional. This option only applies to clustered servers. Separate multiple servers with the ; character.

Mail - Create file replicas in background	Specifies that the file replica's should be created in the background(Yes/No).		Optional.
Address - Internet Address	The internet email address assigned to the new person.		Optional.
ID Info - Public key specification	Specifies the length of the Public key used for the User ID.		
Id Info - Security type USA	The setting determines the encryption strength(Yes/NO).	Yes	Choose either North American (Yes) or International (No). The security type determines the type of ID file created and affects encryption when sending and receiving mail and encrypting data. North American is the stronger of the two types.  This field only is only used when the property "Basics - Create Notes ID" is set to "Yes".
ID Info - Store ID in Domino directory	Specifies whether the the userID should be stored in the Domino Directory.	Yes	

ID Info - Store ID in file	Specifies the location where to store the users ID file.		Users may need access to (a copy of) this file in order to be able to login.
ID Info - ID file name	The file name including the full path to the file that should be created.		In file (default location: <datadirectory>\ids\people\user.id). Click Set ID file to change path.
Groups - Groups	The Groups of which the new person should become a member.		Separate Multiple groups with a ; character.
Roaming - Server name	The server on which the roaming files are to be stored.	servername/tools4ever	Optional. This is the complete name of the server as known within Lotus Notes/Domino environment, not the name of the server as known by the OS hosting the Domino service(s).
Roaming - Replica Servers	Specify the server name to replicate the roaming files to. Seperate multiple server names with ','.	CN=servername/O=tools4ever	Specify the distinguished name of the roaming replica server name! The following format will NOT work: servername/tools4ever'! Use format 'CN=servername/O=tools4ever' instead.

Roaming - Create roaming replica databases in background and	Specifies if the roaming databases must be created in the background by the Domino Administration Process.		If 'Roaming - Create files in background' is set to 'Yes', this option must be set to 'Yes' as well. If the 'Roaming - Create files in background' is set to 'No' or is not specified, you can set this property to 'Yes' and 'No'
Roaming - Subdirectory name	The folder that contains the users roaming information.		Optional. The path must be relative to the server's data directory. The default value is "roaming\<shortname of the person>.
Roaming - Create files in background and	Specifies if the roaming files must be created in the background by the Domino Administration Process.	YES	If set to YES, the existence of any roaming files may not be assumed when calling any subsequent script actions. Note that when set to Yes, the 'Roaming - Create replica databases in background' must be set to Yes as well!
Roaming - Clean-up setting	Specifies the way client side roaming files are cleaned up.		Default value: "Do not clean-up" Possible values: "Clean-up at Notes shutdown" "Clean-up periodically" "Do not clean-up" "Prompt user for clean-up".

Roaming - Clean-up period	Number of days (1-365) after which the roaming users data directory will be removed on the local machine.	Not specified	This setting is used when the clean-up setting is configured as "Clean up periodically". It is ignored otherwise.
Other - Profile name	The name of a R5 user profile to assign.	Not specified	This is generally not used in newer versions of Lotus Notes, as it cannot be used if you are using policies.
Other - location	Departmental or geographical location of the user.		
Other - Organizational unit	The organizational unit of the user.		Optional. A word that distinguishes two users who have the same name and are certified by the same certifier ID.
Other - Comment	A comment about the user, regarding the user's registration.		

Other - Local admin name	The name of a user who has Author access to the Domino Directory but who does not have the UserModifier role. This setting allows the local administrator to edit Person documents.		Optional.
Other - Preferred Language	The language that the user prefers to use		Optional
Limited client flag	Select yes to generate a Lotus Notes person with a 'Lotus Notes Mail' license	Not specified	Optional.
Desktop client flag			

Enforce short name uniqueness flag	Specify "Yes" to enforce uniqueness of the short name.		
Prompt on duplicate mail file	Indicates the action to take on a duplicate mail file.		Select one of the following values "Error on a duplicate mail file (default)" "generate unique mail file name" "replace existing mail file" "skip de person registration".
Prompt on duplicate roaming directory	Indicates the action to take on a duplicate roaming directory.		Select one of the following values: "Error on a duplicate roaming directory(default" "Generate unique roaming directory name" "Skip the person registration".
Prompt on duplicate person	Indicates the action to take on duplicate person .		Select one of the following values "Error on a duplicate person (default)" "Skip de person registration" "Update the existing addressbook registration".
Error if ID file exists	"YES " Specifies that the UMRA script action must fail, and the user must not be created, if the ID file does already exists (Yes/No).	Yes	

Person Document	Specifies the UMRA variable in which should be stored the resulting Person Document created as result of this "Register Person" script action.	%PersonDocument %	Output only. Variable name can be specified on the Out tab only.  The resulting variable can be used in any subsequent script action that requires a person document as input.

**Script Action: Edit person****Function**

Modifies information contained in a specific person document in a Lotus Notes database.

Properties set to "Not specified" will not be modified.

**Deployment**

Typically used as part of a script to modify person properties for existing users, or to specify additional properties after registering a person.

**Properties**



Property Name	Description	Typical setting	Remarks
Person Document	Variable containing an object that represents the Person document to modify.	%PersonDocument% or %NotesDocument%	<p>Mandatory</p> <p>The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.</p> <p>Alternatively, when modifying a person that you have just registered with <i>Script Action: Register person</i> on page 396, a variable containing the person document of the just created person is automatically created by that action.</p> <p>Make sure that the name of the variable specified here as input matches the name of the variable generated .</p>
Basics - First name	The First name of the user.		
Basics - Middle name	The Middle name of the user.		
Basics - Last name	The Last name of the user.		

Basics - User name	The users hierarchical name and other variations.	Not specified	<p>Do not modify this field without a good understanding of the consequences for the Lotus Notes user.</p> <p>Changing the hierarchical name will not change the certifier for the user ID</p> <p>Do not modify this field if you want to move the user in the domino hierarchy, use <i>Script Action: Move person</i> on page 434 instead.</p> <p>separate multiple names by ; character.</p>
Basics - Personal title	The personal title of the person.		For example one of Dr.;Miss.;Mr.Mrs.;Ms.;prof.
Basics - Generational qualifier	The generational qualifier of the person.		For example one of I;II;III;Jr.;Sr.
Basics - Preferred language	The language that the user prefers to use.		
Mail - Mail system	The mail system that is used by this new Person.		For the exact options see the drop down list in the specific action property in the UMRA console application.
Mail - Domain	The mail domain the person is associated with.		

Mail - Mail server	The Name of the Mail Server.		The name of the mail server, as known within Lotus Notes If not specified the same name as the registration server is used.
Mail - Mail file	The name of the mail database file of the person.		Optional. If not specified, the path and file name are set to mail/firstinitial><first7charactersoflastname>.nsf.
Mail - cc:Mail post office	The name of the cc:Mail post office		
Mail - cc:Mail user name			
Mail - cc:Mail location	specifies if the user is local or remote at the cc:Mail post office.		Possible values: "local" "Remote"
Mail - Forwarding address	Address to forward the mail to.		When specifying, Include the domain names for this person. eg. John Smith@Acme@External. Note that the mail will not also be send to the original mailbox when a forwarding address is specified.
Mail - Internet address	The users complete internet address		

Mail - Format preference for incoming mail	The preferred format for incoming mail.		one of "keep in sender's format" "Prefers MIME" "Prefers Notes Rich Text"  Only applies to Lotus Notes, POP, or IMAP mail.
Mail - Encrypt incoming mail	Specifies that incoming mail must be encrypted (Yes/No).		Only applies to Lotus Notes, POP, or IMAP mail.
Real-Time Collaboration - Sametime server	The hierarchical name of the Sametime server.		
Work - Title			
Work - Company			
Work - Department			
Work - Employee ID			
Work - Location			
Work - Manager			
Work - Office phone			
Work - FAX phone			

Work - Cell phone			
Work - Pager number			
Work - Assistant			
Company - Street address			
Company - City			
Company - State/prov ince			
Company - Zip/postal code			
Company - Country			
Company - Office number			
Home - Street address			
Home - City			
Home - State/prov ince			
Home - Zip/postal code			
Home - Country			

Home - Phone			
Home - FAX phone			
Home - Spouse			
Home - Children			
Miscellaneous- Comments			
Miscellaneous- Other X400 address			
Miscellaneous - Calendar domain			
Miscellaneous - Web page			
Miscellaneous - Phonetic name			
Roaming - Clean-up setting	Specifies the way client side roaming files are cleaned up.		Default value: "Do not clean-up" Possible values: "Clean-up at notes shutdown" "Clean-up periodically" "Do not clean-up" "Prompt user for clean-up".

Roaming - Clean-up Interval	Number of days (1-365) after which the roaming users data directory will be removed on the local machine.	not specified	This setting is used when the clean-up setting is configured as "Clean up periodically". It is ignored otherwise.
Administra- tion - Owners	Hierarchical name of the owner of the document. This is the account that has the right to edit this document, if he has Author access to the database.		Usually this is the Same User ID as this very document describes.
Administra- tion - administra- tors	Hierarchical name of users with Author access to the database, but do not have the UserModifier role in the database ACL. If specified here this allows them to edit this document.		You can specify groups, roles(within square brackets[]) and wildcards(for example */sales/acme). Separate multiple entries with commas

Administration - Allow foreign directory synchronization	Allow the users name to be sent to foreign directories(Yes/No).		Enter Yes to allow a users name to be sent to foreign directories; for example a cc:Mail post office directory.The default setting in Lotus Notes is Yes, which means cc:Mail users can lookup Lotus Notes users as if they where cc:Mail users and send mail to them. If you do not want cc:Mail users to send mail to a particular Lotus Notes user, set it to No for that Lotus Notes user.
Password Management - Check password	When set to Yes, the user is required to enter a password to authenticate with servers that have password checking enabled.		
Password management - Required change interval	The number of days at which the user must provide a new password to authenticate		
Password management - Grace period	The number of days after a required change interval in which the users is still allowed to connect with the old password.		



Password management - Change internet password on next login	Force the user to change the Internet password on the next login.		
Policy Management - Assigned policy			
Policy Management - Setup profile(s)			
Client information - Notes client license	The client licenses that this particular user has.		
Ignore empty variable specifications	When set to 'Yes', UMRA tries to identify variables (by checking for %-enclosed names). If a variable is found and the value is empty text or does not exist, the Lotus Notes property is not updated.		

**Script Action: Rename person****Function**

Renames a person, without changing its current certifier

**Deployment**

Typically used as part of a script to manage users and resources in Lotus Notes

**Properties**

Property Name	Description	Typical setting	Remarks
Certifier	Variable containing a object that represents the current Lotus Notes certifier of the person.	%NotesCertifier%	Mandatory Use the action <i>Script Action: Get certifier</i> on page 392 to obtain the certifier object before using it in this action. Make sure that it is the same certifier as used when the person was last registered.

Person document	Variable containing an object that represents the Person document to rename.	%PersonDocument% or %NotesDocument%	<p>Mandatory</p> <p>The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.</p> <p>Alternatively, when modifying a person that you have just registered with <i>Script Action: Register person</i> on page 396, a variable containing the person document of the just created person is automatically created by that action.</p> <p>Make sure that the name of the variable specified here as input matches the name of the variable generated.</p>
First name	The new first name of the user.		
Middle name	The new middle name of the user.		

Last name	The new last name of the user.		
Organizational unit	A short name representing the new user.		Optional. A word that distinguishes two users who have the same name and are certified by the same certifier ID.

**Remarks**

The new hierarchical name of the user generated by Lotus Notes will be:

**<name generated by Lotus Notes from first,middle,last name>/[<Organizational unit>/]<hierarchical name of the certifier>.**

For example, with a certifier called Sales/Tools4ever, the resulting name may be **Mike.G.Smith/ou1/sales/Tools4ever**

**Script Action: Recertify person****Function**

Recertifies a person.

When a UserID is created, it is signed by a certifier. This signature has an expiration date, after which the userID cannot be used anymore to logon. Recertification allows to change this expiration date.

**Deployment**

Typically used as part of a script to manage users and resources in Lotus Notes

**Properties**

Property Name	Description	Typical setting	Remarks
Certifier	Variable containing a object that represents the current Lotus Notes certifier of the person.	%NotesCertifier%	Mandatory Use the action <i>Script Action: Get certifier</i> on page 392 to obtain the certifier object before using it in this action. Make sure that it is the same certifier as used when the person was last registered.

Person document	Variable containing an object that represents the Person document of the person to recertify.	%PersonDocument% or %NotesDocument%	<p>Mandatory, unless the Person name value is specified in which case it may be omitted.</p> <p>The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.</p> <p>Alternatively, when modifying a person that you have just registered with <i>Script Action: Register person</i> on page 396, a variable containing the person document of the just created person is automatically created by that action.</p> <p>Make sure that the name of the variable specified here as input matches the name of the variable generated.</p>
-----------------	---	---	--

**Script Action: Delete person****Function**

Deletes a person and optionally the persons mail files from Lotus Notes.

**Deployment**

Typically used as part of a script to manage users and resources in Lotus Notes

**Properties**

Property Name	Description	Typical setting	Remarks
Person document	Variable containing an object that represents the Person document of the person to delete.	%PersonDocument% or %NotesDocument%	Mandatory.  The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.  Make sure that the name of the variable specified here as input matches the name of the variable generated by the Get Document action
Delete mailfile	Specifies whether the persons mail database must be deleted(Yes/No).		The administration process will create Approve mail file deletion request for the users mail files.

Delete mailfile replicas	Specifies whether the replica's of the persons mailfile must be deleted.		
Delete immediately	Specifies whether the user account must be deleted from the database immediately(Yes/No).		Immediately remove these users name from this domino directory; Administration request will be created to remove their names from acl's, name fields etc.  If NO is specified, all actions will be done by means of administration requests.

**Script Action: Move person****Function**

Moves a person in the Lotus Notes hierarchy by registering the person with an different certifier.

**Important note:** This action can only be used if the person is located in an organization, not if the person is located in an organizational unit. To move a person that is located in an organizational unit, use action Move person (advanced) instead.

**Deployment**

Typically used as part of a script to manage users and resources in Lotus Notes.

**Properties**



Property Name	Description	Typical setting	Remarks
Certifier	Variable containing an object that represents the current Lotus Notes certifier of the person.	%NotesCertifier%	Mandatory Use the action <i>Script Action: Get certifier</i> on page 392 to obtain the certifier object before using it in this action. Make sure that it is the same certifier as used when the person was last registered.
Person document	Variable containing an object that represents the Person document of the person to move.	%PersonDocument% or %NotesDocument%	Mandatory The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.  Make sure that the name of the variable specified here as input matches the name of the variable generated.
New Certifier	Variable containing an object that represents the new Lotus Notes certifier of the person		Mandatory Use the action <i>Script Action: Get certifier</i> on page 392 to obtain the certifier object before using it in this action.

**Remarks:**

If you want to move the person to a organizational unit on which there is no direct certifier, first move the person to the closest certifier above the desired unit in the hierarchy, and then use *Script Action: Rename person* on page 428 to specify the relative unit name.

**Script Action: Move person (advanced)****Function**

Moves a person in the Lotus Notes hierarchy by registering the person with an different certifier. This action is more general, compared to action **Move person**, e.g. a person can be moved from an organization or organizational unit to another organization or organizational unit. With the **Move person** action, a person can only be moved if the person is currently located in an organization.

**Deployment**

Typically used as part of a script to manage users and resources in Lotus Notes.

**Important note:** If the person is registered with the certifier of an organizational unit, the current certifier of the person must be specified by the certifier file of the parent organization of the organizational unit, and not with the certifier file of the organizational unit itself.

See *Lotus Notes example projects* (on page 54) for an example project that uses this action.

**Properties**

Property Name	Description	Typical setting	Remarks
Person document	Variable containing an object that represents the Person document of the person to move.	%NotesDocument%  or  %PersonDocument%	<p>Mandatory</p> <p>The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.</p> <p>Make sure that the name of the variable specified here as input matches the name of the variable generated.</p>

Domino Directory Database	Variable containing an object that represents the Domino Directory Database of the person to be moved. The Domino Directory Database is the Directory Service database of Lotus Notes.	%NotesDatabase%	Mandatory The most general way to obtain the variable is with <i>script action Get Database</i> . on page 449 Make sure that the name of the variable specified here as input matches the name of the variable generated.
Admin Request Database	Variable containing an object that represents the administration request database (admin4.nsf) of the Domino server on which the request is executed.	%AdminRequestDatabase%	Mandatory The most general way to obtain the variable is with <i>script action Get Database</i> . on page 449 Make sure that the name of the variable specified here as input matches the name of the variable generated

Current (parent) certifier file name	The name of the ID file that contains the current certifier of the person. If the person is located directly in an organization, specify the certifier ID file of the organization. <b>Important note: If the person is located in an organizational unit, specify the parent certifier of the organization.</b>	C:\LotusNotes\Ids\cert.id	Mandatory
Password of current (parent) certifier file	The password of the file specified for property <b>Current (parent) certifier file name</b> .		Mandatory Note that the password is stored in an encrypted format.
Target certifier file name	The name of the ID file that contains the certifier of the target organization or organizational unit.	C:\LotusNotes\Ids\TheOu.id	Mandatory

Password of target certifier file	The password of the file specified for property <b>Target certifier file name</b> .		Mandatory Note that the password is stored in an encrypted format.
-----------------------------------	---	--	---

### Script Action: Generate recovery password

**Note:** This action only works with Lotus Notes client software version 7. The action supports all versions of the Lotus Notes Domino server. On computers on which Lotus Notes client software 6.x.x is installed, this action cannot be used.

### Function

Generate a recovery password from a recovery authority. This is the first step to reset the password of an existing ID file.

### Deployment

With special configuration settings, it is possible to reset the password of an Lotus Notes ID file with UMRA actions. The procedure is as follows:

1. The ID-file of the person of which the password must be reset, must be registered using a certifier that contains recovery information. The recovery information consists of a list of recovery authorities, e.g. accounts that can be used to generate recovery passwords.
2. With action **Generate recovery password** a recovery password is generated. Dependent on the certifier recovery information, one or more recovery password are required to reset the password of an ID-file.
3. With action **Recover ID file**, the password is reset using the generated recovery passwords.

Once the password is reset, one can access the Lotus Notes data of the person using the modified ID-file.

In a typical Lotus Notes environment that allows password reset of user ID files, a single recovery authority is used. The ID files are stored in a central location and the name and password of the recovery authority ID are known and specified in UMRA (encrypted). When a user forgets his password, a recovery password is generated using the recovery authority ID, password and the user's ID file. Next, the ID file is recovered.

### Properties

Property Name	Description	Typical setting	Remarks
Recovery authority ID file	The ID file of the person that is specified as a recovery authority for the certifier. A recovery password is generated for this person.		
Recovery authority password	The password of the specified <b>Recovery authority ID file</b> . The password is stored in an encrypted format.		
ID file to recover	The ID file for which the password must be reset.		
Recovery password	The output recovery password, generated by this action.	output: %RecoveryPassword%	

### Script Action: Recover ID file

**Note:** This action only works with Lotus Notes client software version 7. The action supports all versions of the Lotus Notes Domino server. On computers on which Lotus Notes client software 6.x.x is installed, this action cannot be used.

#### Function

Recover an ID file using a recovery password. This is the last step to reset the password of an existing ID file.

#### Deployment

With special configuration settings, it is possible to reset the password of an Lotus Notes ID file with UMRA actions. The procedure is as follows:

1. The ID-file of the person of which the password must be reset, must be registered using a certifier that contains recovery information. The recovery information consists of a list of recovery authorities, e.g. accounts that can be used to generate recovery passwords.
2. With action **Generate recovery password** a recovery password is generated. Dependent on the certifier recovery information, one or more recovery password are required to reset the password of an ID-file.
3. With action **Recover ID file**, the password is reset using the generated recovery passwords.

Once the password is reset, one can access the Lotus Notes data of the person using the modified ID-file.

In a typical Lotus Notes environment that allows password reset of user ID files, a single recovery authority is used. The ID files are stored in a central location and the name and password of the recovery authority ID are known and specified in UMRA (encrypted). When a user forgets his password, a recovery password is generated using the recovery authority ID, password and the user's ID file. Next, the ID file is recovered.

#### Properties



Property Name	Description	Typical setting	Remarks
ID file to recover	The name of the ID file to recover. The ID file identifies the registered person that has forgotten his password.		
Recovery passwords	A single column table that holds all the recovery passwords needed to reset the password.		If only a single recovery password is required, the table only contains a single value.
New password	The new password of the person.		

**Script Action: Set Internet password****Function**

Specifies the Internet password for a person.

**Deployment**

Typically used as part of a script to manage users and resources in Lotus Notes

**Properties**

Property Name	Description	Typical setting	Remarks
Person document	Variable containing an object that represents the Person document of the person to move.	%PersonDocument% or %NotesDocument%	Mandatory  The most general way to obtain the the document variable is with <i>Script Action: Get document</i> on page 454, often in combination with <i>Script Action: Search document</i> on page 464. Make sure that the Document is a correct person document.  Make sure that the name of the variable specified here as input matches the name of the variable generated.
Internet password	The users Internet password. This password is used when accessing the Domino Server via internet protocols such as HTTP, POP3, LDAP or IMAP.		

### Script Action: Set quota

#### Function

This action will set a limit on the size of a Lotus Notes database.

#### Deployment

This action is typically used to set quota's on a users mailbox.

## Properties

Property Name	Description	Typical setting	Remarks
Registration server	The name of the server on which the database is located.		If not specified, the local computer is used (local meaning the computer that executes the script).
Database path/name	The path and filename of the database.		If the property <b>Registration server</b> is specified, the database path is relative to the Domino data directory on the server. Otherwise it should be a fully qualified path name.
Database Quota	The Maximum size in MB that the database is allowed to grow to.		When the value 0 is specified, the database size is not limited by means of a quota.
Warning threshold	The Warning threshold in MB.		A value of 0 implies that no warning will be issued.

## Script Action: Get quota

### Function

This action will retrieve the quota and size information of a Lotus Notes database.

### Deployment

This action is typically used to get quota's from users mailbox.

## Properties

Property Name	Description	Typical setting	Remarks
Database filename	The name of the Lotus Notes database file. If the Domino server is running on the same computer as UMRA, specify the file relative to the Domino data directory, e.g. mail\john.nsf. If the database is maintained on another computer, specify (1) the Domino server name, (2) separator !! followed by (3) the database file path. Example: Server/Domain!!mail\Jonh.nsf.		
Warning threshold	The database size warning threshold in kbytes. This is an output only property.		
Size limit	The database size limit in kbytes. This is an output only property.		
Current size	The current size of the database in kbytes. This is an output only property.		

**Script Action: Configure Out-Of-Office****Function**

Configures the Lotus Notes Out-Of-Office settings of a Lotus Notes person (user).

**Deployment**

Normally the Out-Of-Office settings are controlled by the user himself. When the user is Out-Of-Office, he or she enables the Out-Of-Office agent so that incoming mail messages initiate an automatic response. In special circumstances, for instance when an employee leaves the company, there might be a need to let other people configure Out-Of-Office settings, or to configure Out-Of-Office automatically.

This action assumes that the default Out-Of-Office agent of Lotus Notes is used. The action either enables or disables the out-of-office functions and can also be used to configure the Out-Of-Office parameters.

### Properties

Property Name	Description	Typical setting	Remarks
Mailbox database	The mailbox database of the user for which Out-Of-Office settings are configured.	%NotesDatabase%	Use action <b>Get database</b> to initialize a variable that represents the mailbox database.
Enable/disable Out-Of-Office	A flag to either enable or disable Out-Of-Office settings for t	Yes	Specify 'Yes' to enable and 'No' to disable Out-Of-Office. Even if 'No' is specified, the other properties can be used to update other Out-Of-Office settings.
Person	The distinguished name of the person for which Out-Of-Office settings are configured. By default, this is the owner of the mailbox.	CN=John, O=CompanyOrganization	This parameter is required if Out-Of-Office is enabled and has never been enabled in the past.
Leaving date	The start time and date of the Out-Of-Office period.		The time of this date and time specification is ignored.

Returning date	The end time of the Out-Of-Office period.		The time of this date and time specification is ignored.
Book Busytime	Specify 'Yes' to show in his/her calender that the person is unavailable in the specified period.		
Message subject	The subject of the automatic Out-Of-Office response e-mail.		
Message contents	The contents of the automatic Out-Of-Office response e-mail.		

**Script Action: Process all requests****Function**

Signals the Lotes Notes adminp process to process all outstanding requests

**Deployment**

Typically used in a script to speed up the processing of those actions that result in adminp requests

**Properties**

Property Name	Description	Typical setting	Remarks
Registration server	The name of the server on which the adminp process is located that should process all its outstanding requests.		If not specified, the local computer is used (local meaning the computer that executes the script).

**Remarks**

Several Lotus Notes related script actions result in requests to the Lotus Notes adminp process. This process starts processing these actions at times that are configured in Lotus Notes itself. Often the frequency that the adminp process checks for new actions is quite low, sometimes once an hour or less. This action lets you force the adminp process to check immediately for outstanding request, and process them if possible.

**Script Action: Get database****Function**

Connects to a Lotus Notes database, and creates a variable representing the database.

**Deployment**

Typically used as as first step in the process of querying or altering the database, or the documents it contains.

**Properties**

Property Name	Description	Typical setting	Remarks
Registration server	The name of the server on which the database is located.		If not specified, the local computer is used (local meaning the computer that executes the script).

Database path	The path and filename of the database.		If the property <b>Registration server</b> is specified, the database path is relative to the Domino data directory on the server. Otherwise it should be a fully qualified path name.
Access Directory Service Database	Use the Domino Directory database of the specified server (Yes/No).		If specified, the database path setting is ignored, and a connection to the Domino directory on the server is made (typically this is the names.nsf database on the server).
Database	Variable containing an object that represents the connected Lotus Notes database.	%NotesDatabase%	

### Script Action: Get databases

#### Function

List all databases and/or database templates at a given location.

#### Deployment

Typically used to get a list of available databases, in order to be able select one on which further actions should be performed.

Use *Script Action: Get Database* on page 449 to start operations against a particular database.

#### Properties



Property Name	Description	Typical setting	Remarks
Registration server	The name of the server on which the databases are located.		If not specified, the local computer is used (local meaning the computer that executes the script). Specify the name of the server as known within the Lotus Notes environment.
Databases path	The path to the directory containing all the databases to look for.		optional If the property <b>Registration server</b> is specified, the database path is relative to the Domino data directory on the server. Otherwise it must be a fully qualified path name.  If neither the path nor the registration server is specified, the local Lotus Notes data directory is used.
Options - Get all databases	Specifies whether the names of Lotus Notes databases (*.ns?) must be collected(Yes/No)		
Options - Get all templates	Specifies whether the names of database templates (*.nt?) must be collected(Yes/No)..		

Options - Get all files recursively	Get the files in the specified directory and subdirectories(Yes/No)...		If this flag is not specified, only files in the specified directory are listed.
Databases table	output Variable containing a table that contains the names and related information of all found databases.	%DatabasesTable%	For each found database, there is an entry in the resulting table.

**Format of the resulting databases table**

The resulting table has a list of the found databases or database templates. Each row of the table has the following information:

Column name	Description
Registration server	The name of the server on which the databases is located.
Title	The title of the database.
File name	The file name of the database. for example "names.nsf".
Physical Path	The physical path to the database for example "C:\program files\lotus\domino\data\names.nsf.

**Script Action: Get views****Function**

Create a list of all Lotus Notes views available in a particular Lotus Notes database.

## Deployment

Typically used to be able to select a view name, that later can be used in *Script Action: Get documents* on page 456 to specify which items of the documents should be retrieved.

## Properties

Property Name	Description	Typical setting	Remarks
Registration server	The name of the server on which the databases are located.		If not specified, the local computer is used (local meaning the computer that executes the script). Specify the name of the server as known within the Lotus Notes environment.
Databases path/name	The path to the databases of which the views must be listed.		If the path is relative, and a server is specified, the Domino data directory is used as start folder. If no server is specified, the Lotus Notes data directory is used as start folder. If the database is located elsewhere, use an full path.
Views table	Output variable containing a table that contains the names and related information of all found views in the database.	%ViewTable%	For each found view, there is an entry in the resulting table.

## Format of the resulting databases table

The resulting table has a list of the found views in the specified database. Each row of the table has the following information:

Column name	Description
Registration server (column 0)	The name of the server on which the databases is located.
Database path/name	The path to the database that contains the views.
View/Folder name	The name of the view.
Alias	An alternative name for the view for display purposes.

### Script Action: Get document

#### Function

Retrieves a reference to a Lotus Notes document from a database for subsequent editing

#### Deployment

Typically used in a script that needs to modify the value of fields in a particular arbitrary Lotus Notes document. To specify the new values for fields in the document, use Script Action: Set items(s).

To modify fields in a person document it is more convenient to use *Script Action: Edit person* on page 418

#### Properties

Property Name	Description	Typical setting	Remarks
Database	A variable containing a connected database object.	%NotesDatabase%	This variable is the result of <i>Script Action: Get Database</i> on page 449

Document Note ID	The Note Id of the document to retrieve.	%DocumentID%	<p>This variable is generally the direct result of <i>Script Action: Search document</i> on page 464, which is the recommended way to specify this value.</p> <p>Alternatively, the Note ID of the document can be specified manually. It can be found in the <b>document properties dialog</b> of the specific document as shown for example in the <b>IBM Domino administrator program</b>. The ID is shown on the rightmost tab of this dialog.</p> <p>The value that should be entered here is not the entire ID, but only the decimal value of the last part of the ID.</p> <p>For example, if the note ID ends with NT0000178A , the value that should be entered here is the decimal value of the hexadecimal number 178A, thus 6026</p>
Notes document	Output variable. The resulting reference to a Lotus Notes document is stored in the specified variable.	%NotesDocument%	

**Script Action: Get documents****Function**

Creates a list of all Lotus Notes documents that exist in a particular view or folder in the specified database.

**Deployment**

Typically used to obtain a list of all users in an organization, or comparable overviews.

**Properties**

Property Name	Description	Typical setting	Remarks
Registration server	The name of the server on which the database is located.		If not specified, the local computer is used (local meaning the computer that executes the script, this).  Specify the name of the server as known within the Lotus Notes environment.
Databases path/name	The path to the databases of which the Documents must be listed.		If the path is relative, and a server is specified, the Domino data directory is used as start folder. If no server is specified, the Lotus Notes data directory is used as start folder.  If the database is located elsewhere, use an full path.
View or Folder Name			The name of the view or folder inside the Database. For example '\$Users' or 'People' or 'Server\Servers'.

Documents table	Output variable containing a table that contains the names and related information of all found Documents.	%DocumentsTable%	<p>For each found document, there is an entry in the resulting table.</p> <p>Use one of the standard <i>Script Action: Manage table data</i> on page 528 options to access or manage the contents of the resulting table.</p>
-----------------	--	------------------	---

#### Format of the resulting databases table

The resulting table has a list of the found databases in the specified database. Each row of the table has the following information:

Column number	Column name	Description
0	Document Note ID	The Lotus Notes ID of the document. This is the value that can be used in a <i>Script Action: Get document</i> on page 454 to access the specific document.
1	Registration server	The name of the server on which the databases is located.
2	Database path/name	The path to the database that contains the views.
3	View/Folder name	The name of the view.
4 5 ...	<names by Lotus Notes>	A variable number of columns that are defined in the specific Lotus Notes view itself.

**Script Action: Create document****Function**

Creates a new Lotus Notes document at the specified location in the Lotus Notes database.

Returns a reference to the just created Lotus Notes document to allow subsequent editing.

**Deployment**

Typically used in a script that needs to create new Lotus Notes documents. To specify the values for fields in the document, use Script Action: Set items(s) afterwards.

To create a person document that should represent a valid registered Lotus Notes user use *Script Action: Register person* on page 396 instead.

**Properties**

Property Name	Description	Typical setting	Remarks
Database	A variable containing a connected database object.	%NotesDatabase%	This variable is the result of <i>Script Action: Get Database</i> on page 449
Form Name	The name of the Lotus Notes Form used for the new document.		If not specified the default form of the database will be used. The form defines for instance which fields (items) can be used for the document and their default values.
Folder Name	The name of the folder where the document should be created.		This is a virtual "folder" inside the Lotus Notes database, not a folder on the file system



Copy Note document	A reference to a existing note document.	%SourceDocument%	Optional argument. If specified, the new document is a copy of the existing document  The required reference to the source document can be retrieved by using for example <i>Script Action: Get document</i> on page 454.
Note document	Output variable. The resulting reference to the created Notes document is stored in the specified variable.	%NoteDocument%	

**Script Action: Copy document****Function**

Copies an existing Lotus Notes document from one database to another Lotus Notes database. Returns the ID of the new copy of the document.

**Deployment**

Typically used in a script to copy documents between Lotus Notes databases.

**Properties**

Property Name	Description	Typical setting	Remarks
Source database	A variable containing a connected database object. The database must contain the document to be copied, identified by property NotesID.	%NotesDatabase%	This variable is the result of <i>Script Action: Get Database</i> on page 449
Destination database	A variable containing a connected database object. The document is copied and stored in this database.	%DestinationDatabase%	This variable is the result of <i>Script Action: Get Database</i> on page 449
NotesID	The ID of the document to copy. The ID is the result of UMRA Lotus Notes action 'Search documents'. Example: 5318.		The document must exist in the Lotus Notes database that is specified with property Source database.
Result NotesID	The resulting ID of the copy of the document. This is an output only property.	%CopyNotesID%	

### Script Action: Delete document

#### Function

Deletes the specified document from its database

## Deployment

Used to delete a document from a Lotus Notes database. This is a low level action, so do not use this to delete for instance a registered user. To delete a user use *Script Action: Delete person* on page 432 instead.

## Properties

Property Name	Description	Typical setting	Remarks
Notes document	reference to the Lotus Notes document that should be deleted.	%NotesDocument%	The required reference to the document can be retrieved by using for example <i>Script Action: Get document</i> on page 454.  Important: After this script action the reference to the document is no longer valid, so the variable should not be used anymore further in the script.
Notes database	A data structure representing the Lotus Notes database that contains the note. Specify either this property and property 'Note ID' or specify single 'Notes document'.		See remarks

Note ID	The NoteID of the note that must be deleted. Specify either this property and property 'Notes database' or specify single property 'Notes document'.		See remarks
---------	--	--	-------------

### Remarks

To delete a document, the input of this action can be specified in 2 ways:

1. **Notes document**  
By using script action **Get document**, the required reference is obtained. In this case, the properties **Notes database** and **Note ID** should not be specified.
2. **Notes database and Note ID**  
If the **Notes document** variable is not available this method can be used. In this case, the property **Notes document** should not be used. The property **Notes database** property can be obtained using action **Get database**. The Note ID can be obtained in various ways.

### Script Action: Get item

#### Function

Retrieves the current contents of a specific item (field) of a Lotus Notes document

#### Properties

Property Name	Description	Typical setting	Remarks

Notes document	reference to the Lotus Notes document that should be deleted.	%NotesDocument%	The required reference to the document can be retrieved by using for example <i>Script Action: Get document</i> on page 454.
Item name	The name of the item of the document		
Item type	The type of the value of the item		The value read from the Lotus Notes document is converted to the specified type. If the conversion is not possible , an error can be generated. If the property is not specified a default conversion method is used.
Error if not found	Generate an error if the specified item is not found in the document	YES	
Error if conversion fails	Generate an error if the conversion fails	YES	If NO, the item value is converted to the best fitting type if possible. This may give errors further in the script if a specific type is required in an other action.
Item value Variable	Generate an error if the specified item is not found in the document	YES	Output only. This variable will contain the value of the item.

**Script Action: Get item size****Function**

Retrieves the size in bytes of a specific item of a Lotus Notes document.

**Deployment**

Typically used to determine the size of an item to check if the maximum size (32k or 64k bytes) is not exceeded if the item is updated.

**Properties**

Property Name	Description	Typical setting
Document	A data structure representing the Lotus Notes document that contains the item for which the value size must be determined. The property is initialized with Lotus Notes action 'Get document'.	%NotesDocument%
Item	The name of the Lotus Notes document item of which the size must be determined. The size of the item is stored in bytes. Example: Department	
Item size	The name of the variable in which the size of the Lotus Notes document item is stored. The size is stored in bytes. This property is an 'output only' property.	%ItemSize% (output)

**Script Action: Search documents****Function**

Searches for documents in a specified Lotus Notes database that match certain criteria. It returns a list of Note ID's of the documents that satisfy the criteria.

### Deployment

Typically used to retrieve the Note ID of a particular document that must be modified. The retrieved Document ID is subsequently used in *Script Action: Get document* on page 454 to get a reference to the document which is needed in actions to modify it.

### Properties

Property Name	Description	Typical setting	Remarks
Database	A variable containing a connected database object which must be searched.	%NotesDatabase%	This variable is the result of <i>Script Action: Get Database</i> on page 449
Search formula	The formula that defines the search criteria.		Example: @LowerCase(Lastname)="smith".
Begin date-time	Documents last modified before the specified data-time value are ignored.		If not specified this criterion is not used (no documents are excluded)  If specified in a variable, the variable must be of the <b>UMRA date-time</b> type, not a text variable.

End date-time	Documents last modified after the specified data-time value are ignored.		<p>If not specified this criterion is not used (no documents are excluded)</p> <p>If specified in a variable, the variable must be of the UMRA date-time type, not a text variable.</p>
Maximum Count	<p>Output variable.</p> <p>Maximum number of document ID's to return.</p>	not specified	If not specified, all documents are returned.
Document Result table	<p>Output variable.</p> <p>A table containing the list of Note ID's of the matching documents.</p>	%DocumentIDTable %	The table holds a single column called "DocumentID".

### Script Action: Query Document Items

#### Function

Searches for documents in a specified Lotus Notes database that match certain criteria. It returns a table containing the value of items from the documents that satisfy the criteria. Each document that satisfies the criteria corresponds with a row in the table. The items that are returned for each document are configured in this script action

#### Deployment

Typically used to do a query to a Lotus Notes database, and put the results in a table. It is very similar to the action *Script Action: Get*



*documents* on page 456. The difference is that that action uses an predefined view in the database, and this action uses a general query.

### Properties

Property Name	Description	Typical setting	Remarks
Database Variable	This variable contains a connected database object. This database contains the documents that are queried.		This variable is the result of <i>Script Action: Get Database</i> on page 449.
Output table Variable	The name of the variable that contains the resulting table	%ItemTable%	Required, output only For each found document, there is a row in the resulting table; for each item in the item list, there is a corresponding column with the value of that item. The items names from the Item list are copied to the column headers. Use one of the standard <i>Script Action: Manage table data</i> on page 528 options to access or manage the contents of the resulting table.
Search formula	The formula that defines the search criteria.		Example: @LowerCase(Lastname)="smith".

Documents items	A list of Items whose values will be stored in the resulting table		<p>For each item in the item list, there is column with the value of that item in the resulting table.</p> <p>The first Item is always the NoteID of the document.</p> <p>For each item can be specified what should be done if it does not found in the document</p>
-----------------	--	--	---

### Script Action: Sign/Unsign document

#### Function

Signs or unsigns the specified document. The Lotus Notes Account on whose behalf the signing occurs, is the account specified in the notes.ini file used for the complete Notes session .

#### Deployment

For instance used to create documents that are used by processes that require signed document, or in order to alter certain fields of documents that are signed.

#### Properties

Property Name	Description	Typical setting	Remarks
Notes document	A reference to the Lotus Notes document to handle	%NotesDocument%	The required reference to the document can be retrieved by using for example <i>Script Action: Get document</i> on page 454.

Sign Document	Signs the document(Yes/No)		<p>When a document is signed this guaranties to other users that those document items(fields) that have the "signed" property, are not altered after the signing of the document.</p> <p>Some processes that read Documents may require a document to be signed before they consider it valid, for reasons of security.</p>
Unsign Document	Unsigns the document(Yes/No)		<p>Removes a signature from a document. This is required before items of the document Fields that have the "signed" property can be modified.</p>

### Script Action: Set item(s)

#### Function

Creates or modifies values of specified Document fields (note items).

#### Deployment

Used to programmatically edit a document. General action to edit existing (or just created) documents in a Lotus Notes database.

When there is a specialized script action available for the document you want to modify, you are strongly advised to use that one instead. For instance, to edit person documents, use *Script Action: Edit person* on page 418 instead. The "Set item(s)" action is a very general document modification action. It is able to create or modify virtually any field of any document. Due to its general nature however, the action cannot

and does not generally check whether the resulting document is fit for any particular purpose. It is the responsibility of the script designer to specify the correct fields that result in a valid Lotus Notes document.

See *Lotus Notes example projects* (on page 54) for an example project that uses this action.

### Properties

Property name	Description	Typical setting	Remarks
Notes Document	A variable containing a reference to a Lotus Notes Document	%NotesDocument%	This variable is typically the result of <i>Script Action: Get document</i> on page 454
Number of document items	This Property represents a list of items, each item containing a command to set or modify a specific field of the Document.	Value is specified by creating the list of items in the action dialog	See Item Properties below for the possible settings and options for each item.

### Item properties

Item property name	Description	Typical setting	Remarks
General: Item name	The name of the field to create or modify		

General: Item Type	The value-type of the field. (text, text list, date-time, numeric etc.)	text	
General: Options	Specifies what to do if there already is a field with the specified item name in the current document		<p>Possible options:</p> <ul style="list-style-type: none"><li>▪ Error if exist. The entire script action will not be performed and an error will be generated, if any of the fields with this setting already exist in the document</li><li>▪ Delete existing first. If the current document already contains this field, the entire field is removed before the new field is added.</li><li>▪ Append if exist. The new value will be merged with the existing value as specified in the value options. If there are no special options, the value will be appended at the end of the current one.</li></ul>

General: Item creation flags	Several flags that determine specific Lotus Notes setting regarding the field		Possible options: <ul style="list-style-type: none"> <li>▪ sign</li> <li>▪ encrypted</li> <li>▪ protected</li> <li>▪ names</li> <li>▪ readers</li> <li>▪ readers-writers</li> <li>▪ placeholder</li> <li>▪ summary</li> <li>▪ auto-summary</li> <li>▪ unchanged</li> </ul>
Value:	The value that the field should get		Dependent on the Item type there may be special options. see below.

**Item property: Item creation flags**

**Sign:** Items where this flag is set will be sealed when the document is signed, for instance with *Script Action: Sign document* on page 468.

**Encrypted:** Items where this flag is set will be encrypted, when the document itself is encrypted. Fields without this flag will not be encrypted.

**Protected:** Editor access is required to change the item.

**Names:** The item is a text field that contains a list of users or groups. often used together with the "readers" or "Authors" option.

**Readers:** The item is a item containing a list of readers, used for access control. The "names" option must also be specified if this option is specified.

**Authors:** The item is a item containing a list of authors, used for access control. the "names" option must also be specified if this option is specified.

**Placeholder:** The item is a placeholder field.

**Summary:** The item added to the document, and is also placed in the summary buffer of Lotus Notes. This is required for the item to be visible in any view. If the item is larger than 32 k it does not fit in the summary buffer and an error is generated. Use this setting if it is required that the particular field is always visible in views, or if you need to know that it cannot be shown. Your UMRA script may then react on the error situation either by making sure that the value is smaller than 32 k, add it without this flag, or perform some other required action. Only specify this flag if you really need to know if an item does not fit.

**Auto-summary:** The item is added to Lotus Lotus Notes, and if it is smaller than 32 k it is also placed in the summary buffer which is required for it to be visible in any view. No error is reported if it is larger than 32 k. By default this is on. If you require a notification if an item does not fit, use the "summary" instead.

#### **Item Value**

The value of an item is specified in on the value tab. The available options depend on the specified Item Type on the general tab.

#### **Item type: Text**

Value property name	Description	Typical setting	Remarks
Text	The exact text value of the resulting field.		If the general option "Merge if exist" is specified the text is appended to the existing value of the item.

**Item type: Text List**

<b>Value property name</b>	<b>Description</b>	<b>Typical setting</b>	<b>Remarks</b>
Operations	Specification how to merge the new text values with the current one. Options are:  Set (unconditionally replace existing values with specified values)  Append values(s)  Insert value(s) at begin  Remove (no error if not found)  Remove (error if not found)	Set	
Text item values	A list of new text values.		

**Item type: Date-time**



Value property name	Description	Typical setting	Remarks
Date time value specification	The date-time value to set the field to		if Specified by a variable, it should be a UMRA Date-time type variable.
Date time operation	<p>Specifies how to merge the item with existing values. There are 3 options.</p> <p>1) Set item value to the specified date-time value.</p> <p>This results in a single date-time as specified</p> <p>2) Append the specified date-time value to the current values</p> <p>Any existing list of date-time values is extended with the new value.</p> <p>3) Insert the specified date-time value at the beginning of the current list of date-time values.</p> <p>Any existing list of data-time values is retained, and the new value is added in front of the current values.</p>		

**Item type: Numeric**

Value property name	Description	Typical setting	Remarks
Number value specification	The numeric value to set the field to		if specified by means of a variable, the result must be resolvable to a numeric value. If a variable is used it is therefore best to specify only a single UMRA variable of the numeric type.

Number operation	<p>Specifies how to merge the item with existing values. There are 3 options.</p> <p>1) Set item value to the specified number value. (default)</p> <p>2) Append the specified numeric value to the current values</p> <p>Any existing list of numeric values is extended with the new value.</p> <p>3) Insert the specified numeric value at the beginning of the current list of numeric values.</p> <p>Any existing list of numeric values is retained, and the new value is added in front of the current values.</p>		
------------------	---	--	--

**Script Action: Delete Item****Function**

Deletes specified Document field (note item) from a document

**Deployment**

Used to edit a document.

**Properties**

Property name	Description	Typical setting	Remarks
Notes Document	A variable containing a reference to the Lotus Notes document to edit.	%NotesDatabase%	This variable is typically the result of <i>Script Action: Get document</i> on page 454
Item name	The name of the field to delete		

### Script Action: Update profile document

#### Function

Sets the text, text list, or numerical value of an item of a Lotus Notes profile document.

#### Deployment

Profile documents are typically used to store application and user preference data in order to facilitate personalization. These documents are like typical Domino database documents, except they are excluded from the database document count and are cached when the database is opened. This action updates a specific text item of a Lotus Notes profile document. The action cannot be used to update other types of data.

See *Lotus Notes example projects* (on page 54) for an example project that uses this action.

#### Properties

Property Name	Description	Typical setting	Remarks
Database	A data structure representing the Lotus Notes database that contains the profile document. The property is initialized with Lotus Notes action 'Get database'.	%NotesDocument%	The required reference to the database can be retrieved by using <i>Script Action: Get Database</i> on page 449.
Profile name	The name of the profile document to be updated. Example: 'CalendarProfile'.		
Field name	The name of the field of which the value must be updated.		To set the value of a profile document item, the <b>Field name</b> must be specified. If <b>the Field name</b> is not specified, the document will be signed if the <b>Sign flag</b> is set.

Field value	The new value of the specified profile document field.		Lotus Notes supports various types of profile field item values. By default and for most fields, text values are used. See the <b>Remarks</b> section for other supported types and how to specify these types. When the value [ <b>*delete*</b> ] is specified, the field value is not stored but the field itself is deleted from the profile document.
Field item flags	Optional: The flags that define the characteristics of the field item. Add the following numbers to determine the exact value: 1=sign, 2=seal, 4=summary, 32=readwriters, 64=names, 256=placeholder, 512=protected, 1024=readers, 4096=unchanged.		Example: to set the flags 'sign' and 'summary', specify a value of $1 + 4 = 5$ .
Sign flag	Optional: Specify 'Yes' to sign the profile document when changes are applied.		See the <b>Remarks</b> section for additional information on signing profile document without updating item fields.

Signature time field	Optional: Specify the name of the field that should contain the date and time of the signature of the profile document.	[not specified] or SignatureTime	In most cases, this field is not specified. See the <b>Remarks</b> section for more information.

### Remarks

By default, the values set are text values (single item text values). It is also possible to specify the value as a Lotus Notes **number value** and a Lotus Notes **text list value**. To do so, in UMRA, specify a variable for the **Field value** with the corresponding type, as shown in the following table.

Lotus Notes field item value type	UMRA variable type	Example Field value specification
TYPE_TEXT (simple text, default type)	Field value specified as text -or- Field value specified using a variable of all types not used for	Archived Emails %ArchivedEmailsProfileName%
TYPE_NUMBER (numeric value)	Field value specified as a single variable of UMRA type <b>numeric</b>	%Number5%
TYPE_TEXT_LIST (array of text values)	Field value specified as a single variable of UMRA type <b>text list</b>	%ArchivePrivatePolicyList%

### Delete profile document item field

To delete an item from the profile document, specified field value **[\*delete\*]**. This will delete the specified item from the profile document.

### Sign a profile document

To sign a profile document without changing any of the profile document item fields, do not specify a **Field name**. Set the **Sign flag** to **Yes**. This will sign the profile document.

### Signature time field

For typical profile documents, it is required to add a field to the profile document that contains the date and time of the signature of the profile document. An example is the **archive profile** document, part of the user's mail database. To support this function, set the **Sign flag** to **Yes**, e.g. sign the document and specify the name of the field that must contain the time and date of the signature. When the document is signed, the date and time value is retrieved from the document and added to the document. If the **Sign flag** is not specified or set to **No**, this field has no effect.

### Example

This action is used to specify the archive settings of a Lotus Notes database. In the Domino administrator, this corresponds with the following action: Select a Lotus Notes database (.nsf file), right click and select **Properties**. Click **Archive Settings** and select **Advanced**. Check the option: **Log all archiving activity into a log database** and specify the database. To use the UMRA action **Update profile document** to do the same, specify the following properties.

Property Name	Value	Example
Database	The UMRA variable obtained with action 'Get database' to access the Lotus Notes database.	%NotesDatabase%
Profile name	archive profile	archive profile
Field name	ArchiveLogDBPath	ArchiveLogDBPath
Field value	The name of the log database.	archive\log_user123.nsf



Field item flags	5	5
Sign flag	Yes	Yes

**Script Action: Update ACL****Function**

Creates or modifies an **Access Control Entry** in the Access Control List of a Notes Database

**Deployment**

Used to edit the allowed access of a specific person to a Notes Database.

**Properties**

Property name	Description	Typical setting	Remarks
Database variable	A variable that contains the database of which the access should be modified.		This variable is the result of <i>Script Action: Get Database</i> on page 449
ACE name	The name of the Lotus Notes user for which the security is modified, for instance the short name of the user.		Only one user can be specified. To change the access for more users, use this action more times in the UMRA script with different users. The user must be specified using the following notation: CN=name/O=org

Update the ACE	If selected, the ACE will be created or modified according to the specifications		the security for the user will be set according to the specifications
Delete the ACE	If selected, the ACE will be removed from the ACL		The specific security settings for the user will be removed from the ACL. The user may still have access due to group memberships.
User Type	The type of user represented by the ACE name	A choice of <ul style="list-style-type: none"><li>▪ Person</li><li>▪ Server</li><li>▪ Mixed group</li><li>▪ Person group</li><li>▪ server group</li><li>▪ unspecified</li></ul>	This is mainly used for display purposes
Access	The main access category of the user	A choice of <ul style="list-style-type: none"><li>▪ Manager</li><li>▪ Designer</li><li>▪ Editor</li><li>▪ Author</li><li>▪ Reader</li><li>▪ Depositor</li><li>▪ No Access</li></ul>	

Privileges	De detail level privileges	A combination of <ul style="list-style-type: none"> <li>▪ Create documents</li> <li>▪ Delete documents</li> <li>▪ Create private agents</li> <li>▪ Create personal folders/views</li> <li>▪ Create shared folders/views</li> <li>▪ create LotusScript/Java agents</li> <li>▪ Read public documents</li> <li>▪ Write public documents</li> <li>▪ Replicate of copy documents</li> </ul>	Depending on the access category, some privileges are preset.
------------	----------------------------	--	---

### Script Action: Execute agent script

#### Function

Creates, compiles, executes and deletes a Lotus Notes agent in an existing Lotus Notes database. The agent consists of an configurable Lotus Notes script. The action uses an existing database and executes the following procedure:

1. Create an agent in the database. The action specifies the name of the agent. Before the agent is created a new temporary Lotus Notes document is created to hold the agent;
2. The Lotus script of the agent is set. The Lotus script text is completely configurable and specified by the UMRA action. Next, the script is 'compiled' in Lotus Notes. The result is stored in the agent and ready for execution. The target document(s) of the agent must be specified and accessed as part of the Lotus script.
3. The agent is executed (optional).
4. The agent and the Notes document that holds the agent are deleted (optional)

## Deployment

Typically used to execute Lotus Notes tasks that can run only as Lotus script in Lotus Notes agents. The action can be used for instance to execute certain tasks by using the administration process database. The administration process is used in a lot of confirmation operations, for example to confirm the deletion of a user's mail file when the user is deleted. Once the administration request to confirm the mail file deletion exists, the request must be confirmed to complete the operation and delete the mail file. This must be done manually by an administrator. To automate this process, an agent can be created in the administration process database. The agent contains Lotus script code and selects the request document and confirms the request using a script library of the administration process database. The UMRA project to execute this procedure can be found at `.\Example Projects\LotusNotes\LotusNotesApproveMailfileDeletion.xml`.

## Properties

Property Name	Description	Typical setting
Database	A data structure representing the Lotus Notes database that is used to execute the agent.	%NotesDatabase%
Agent name	The name of the temporary agent Lotus Notes agent. Example: UMRA Lotus Notes agent. The name does not have to be unique.	
Agent script	The Lotus script of the agent. The Lotus script text is immediately executed when the agent is executed. The script itself should select the appropriate Lotus Notes database documents.	
Agent comment	A free text describing the agent.	

Run agent flag	Optional: A flag indicating if the agent should be executed immediately when created. If not specified, the agent is executed when created. If set to 'No' the agent is created but not executed.	
Delete agent flag	Optional: A flag indicating if the agent should be deleted when created and (optionally) executed. If not specified, the agent is deleted. If set to 'No' the agent is not deleted.	
Note ID	Optional: An output value, the NoteID of the note that holds the created agent. This property is an 'output only' property and is generated automatically. This property is to be used in other script actions. Store this value in a variable in order for instance to be able to delete the agent.	%AgentNoteID%

### Example script

The following script contains the Lotus script code of the example project. In UMRA, the variable %NoteID% refers to the administration process request document to confirm the deletion of a mail file. The syntax of the %NoteID% variable is as specified by Lotus Notes, for example: 00001E40. (eight characters, hexadecimal notation with leading zero's).

```
Option Public
Option Declare
%INCLUDE "Isconst.lss"
%INCLUDE "Isxbeerr.lss"
%INCLUDE "Iserr.lss"
Use "AdminRequestLib"

Sub Initialize
Set s = New NotesSession
Set db = s.CurrentDatabase
Dim doc As NotesDocument
```

```
Set doc = db.GetDocumentByID("%NoteID%")
If Not(doc Is Nothing) Then
Call ApproveRequest(doc)
End If
End Sub
```

The agent creates a notes session `s` and selects the request document `doc` by specifying the ID of the document. Once found, then the subroutine **ApproveRequest** of database library "**AdminRequestLib**" is called.

#### **4.3.8. SAP actions**

UMRA supports over 30 actions to manage the SAP environment, accounts and related resources. To configure UMRA to support the UMRA SAP actions, see *UMRA and SAP*. For more information on the individual actions, you are referred to the descriptions of the action and action attributes.

#### **SAP actions**

UMRA supports over 30 actions to manage the SAP environment, accounts and related resources. To configure UMRA to support the UMRA SAP actions, see *UMRA and SAP*. For more information on the individual actions, you are referred to the descriptions of the action and action attributes.

#### **4.3.9. TOPdesk**

TOPdesk is a help desk system with the ability to manage incidents at several levels. Connect incidents to users, sites etc. UMRA supports TOPdesk thru its URL interface. This means that UMRA can do anything in TOPdesk, which can be done thru the url interface. The URLs send to TOPdesk are described in a PDF document which can be requested from the TOPdesk support desk. The URLs are created and modified by several UMRA actions. UMRA has also an 'TOPdesk Invoke action URL'

action to send the URL to TOPdesk. This is sophisticated action which can

perform HTML analysis to catch error messages and fetch TOPdesk IDs.

Some URLs are much used (For example to create an incident.), those URLs have special actions in UMRA.

The following URLs have special actions in UMRA:

- \* Create incident.
- \* Create Person
- \* Create Site

The URL objects generated by this action can be modified by the normal TOPdesk UMRA actions to add change or remove fields. The generated URL object can be invoked by the 'TOPdesk Invoke action URL' action, like any TOPdesk URL object.

also UMRA provides the following actions to retrieve information from the TOPdesk system:

- \* Get Person
- \* Get Incident
- \* Get Persons
- \* Get Unid
- \* Get Unid list

Those actions do not generate an URL object, but just a table or other output variable.

The passwords used in the URLs are never shown in the logs. Make sure the connection to TOPdesk is using https, because the passwords are part of the URL and therefore not encrypted when sent to TOPdesk. If https is not used those URLs can easily be monitored. By using https this monitoring is not possible.

The debug option should only be used when instructed to do so by a Tools4ever employee. The debug option will generate lots of extra log information. However passwords are still not logged.

#### **4.3.10. Education**

UMRA supports a number of **connectors** for educational systems. For each of these systems, a collection of UMRA actions is available. Each action implements a specific task for the system. The connectors consists of the specific UMRA actions for the system.

##### **Aura**

Aura is a Dutch company that offers library software that is primarily used in schools. The UMRA connector for Aura integrates the student information system, for instance Magister, @VO or nOISe, with the Aura software. Changes in the student information system are automatically propagated to Aura.

To setup the UMRA connector for UMRA, see *Aura connector installation* for more information.

##### **Aura actions**

##### **Aura Setup connection**

Use this action to build a connection to the **UMRA-Aura-Webservice** component. Provide a PowerShell session, so the connection can be used by following Aura actions.

##### **Aura Initiate all users**

When first installed UMRA is not able to modify existing Aura users. Every existing Aura user has to be initialized first. Use the action 'Aura



Initiate all users' to enable modification of existing Aura users. Users created by UMRA do not have to be initialized.

**Aura Get Users**

This action retrieves all users and all their properties from the Aura system. Users not initialized and not created by UMRA are also included.

**Aura Create User**

To create a new user in the Aura system, this action must be used. Depending on the setting of the 'PasnummerIsLenersCode' key in the web.config file of the **UMRA-Aura-Webservice** the rental code will be used or ignored. If the card number is set to be equal to the rental code, the value of the rental code will be ignored by Aura.

**Aura Get user exists**

Checks if the specified user exists in the Aura system. Aura uses the 'card number' for identification of users. This should be the same as the student id in the student administration system.

**Aura Edit user**

To update existing user, this action must be used. The card number identifies the user in Aura. Empty values will not be changed.

**Aura Delete user attribute**

This action clears an attribute from an Aura user.

**Aura Get user info**

To retrieve all information from an existing Aura user. (Only UMRA created or initialized users.)

### **N@tSchool**

The UMRA N@TSchool connection is a very advanced connector based on SOAP. The SOAP webservice is part of the N@tSchool software and no special requirements to setup the connector apply. The UMRA Powershell Agent service is used to access the webservice.

To create a fast and reliable connection a caching system is build into this connector. To use the advantages of this caching mechanism, the order of execution of the actions is important (although for stability the order has no consequences). A few understandings of how N@TSchool works, will make the implementation of the actions easier.

### **Containers**

N@TSchool users are divided in containers. Those containers are the groups at the highest level in the N@TSchool interface. Because loading items from a container is time-consuming the containers that UMRA is operating on, can be limited. It is possible to exclude specific containers or just to include only a few. When excluding containers, all other containers (including containers created at a later point) will be included in the search. When including containers, only those containers will be included in the search operations. Containers that are created later have to be included explicitly. Excluded containers take priority over included containers. So if an included container is later on excluded by the **N@TSchool exclude root containers** action, this container will not be included. The special containers **Everyone**, **Users** and **Administrators** will not be included in searches. The result of the include and exclude root container actions will be called the **container set**. To retrieve the id's of the containers check the UMRA log. The id's are send to the log when the **N@TSchool Setup connection** action is executed.

### **Caching**

To improve performance of the N@TSchool connector a caching mechanism is implemented. Understanding of this mechanism is important to get the best performance of the N@TSchool actions.

There are 3 caches.

1. **User cache:** This cache contains all the user, group and membership information of all the included containers, without the excluded containers.
2. **Changes cache:** This cache contains all the modifications UMRA has to make.
3. **User name cache:** This cache contains all the user names of all N@TSchool users in all containers.

When creating a connection, all user names are loaded by a background thread into a cache. This cache is used when the **N@TSchool Get user name available** action is run. Because this cache is filled by a background thread, the **N@TSchool Setup connection** action will return immediately. As soon as this cache is used, the action will wait until it's filled. Therefore it can be wise to create a N@TSchool connection as soon as possible in the UMRA script and first perform all other tasks before calling the **N@TSchool Get user name available** action.

When requesting user information (for example by the action **N@TSchool Get all users**), the user is looked up in the changes cache and all containers in the 'container set'. When the users in a container set are not yet loaded in the cache, the users are requested from N@TSchool and saved to the cache. After the cache is filled the requested information is retrieved from the cache and send back to UMRA. Therefore the first time a single user is looked up, the whole container cache is filled, till the user is found. If the user does not exists, this can take some time. The next call however will return immediately. When the members of a group are requested the whole container set is scanned for members and every container is loaded in cache if not already done so.

When users, groups and/or memberships are created, removed, changed, these changes are recorded in the changes cache. As soon as the **N@TSchool Process changes** request is called, all the changes are sent to N@TSchool and all the caches are cleared, also the user name cache will be reloaded in background. Therefore It's wise to perform a **N@TSchool Get user name available** as late as possible after a **N@TSchool Process changes**. Also do not perform a **N@TSchool Process changes** after every modification but for example after 20 modifications. A modification will just update the changes cache and reload the user cache. Therefore those actions will not take long to process even if the caches are empty.

#### **Process changes**

Because all modifications are saved to a cache, the actions to perform those modifications will not return an error in case of an illegal modification, because the modification is not yet performed. The **N@TSchool Process changes** will send back reporting variables. Use those to check if ALL modifications are processed correctly.

#### **TeleTOP**

TeleTOP is a LMS (Learning Management System) used by a large number of dutch schools. With the TeleTOP actions in UMRA it is possible to create users and courses inside TeleTOP.

The connection is SOAP based. TeleTOP can be configured to use https, this is the preferred configuration for the UMRA connector. HTTP connections will work, but are not supported by Tools4ever. The data communicated with TeleTOP is sensitive and therefore HTTPS should be used.

If https is not enabled in TeleTOP, request the TeleTOP support desk to enable https. After https support is enabled by TeleTOP change the TeleTOP configuration to enable https.

Open the Learning Management System (LMS) in TeleTOP. Use the URL that shows in your browser of the TeleTOP Learning Management System as the value for the TeleTOP system in the 'TeleTOP Setup connection' action.

### **Users and courses**

TeleTOP exists of users and courses. A course holds study materials (such as readers and other interesting stuff.) UMRA does not maintain the contents of a course, teachers do. In TeleTOP any user can be a teacher in any course. Teacher or student is just the role a user has in a course. So from a user account perspective there is no difference between a student and a teacher, but from a course perspective there is. When adding users to a course the role has to be specified. The role defines whether a user is a teacher, student or a guest in the course. Depending on the role in the course (Teacher, Student, Guest) the user has rights in a course. The rights of every role are influenced by the overall TeleTOP settings as well as by the course settings.

### **Courses**

Courses are assigned to a year. In TeleTOP years are defined as a system variable named 'Years'. Courses cannot be assigned to a year that does not exist in the system variable 'Years'. In the current version of TeleTOP it is not possible to set or change the year of the course. Courses are always created in the current year. Therefore the system variable 'Years' must contain a value for the current year. The values of the 'Years' variable are specified in the form 'label|YY'. The label normally is the 4 digit year, but can be anything. After the pipe a 2 digit year code must be specified. For example, to be able to create courses in the year 2010, the system variable 'Years' must contain a value '2010|10'. The system variable 'Years' cannot be managed by UMRA. Use the web interface to manage this variable. A course can have 3 extra properties known as course keys. The values of those properties are multi-valued. Mostly the interpretation is hierarchical, although it is not mandatory to interpret them as hierarchical values. For example the value of 'coursekey2' is mostly a subvalue of 'coursekey1'. So if 'coursekey1' contains the name of the school, 'coursekey2' could contain

the other (for example, informatica). This can be handy to filter courses for a school or differentiation. Course keys are, just like years, system variables. Unlike years, course keys can be created and removed by UMRA. Also UMRA can add course key settings to, or remove course key settings from a course. Note: After creating a course, it will take a while before the web-pages of the course, through which teachers manage the course materials and students log-in, are available. Depending on the internal system settings at the TeleTOP location, it will take up to a day, but mostly a few minutes, before the web-interface is available. Note: Courses are, currently, limited to hold only 800 users. Although older courses may have already more than 800 users assigned, TeleTOP will not return the members of courses with 800+ users. See preparations for a work around.

## Users

Users can be created, edited and disabled by UMRA, but not removed. Users are identified by their 'User ID'. This value must be unique for every user in the TeleTOP environment. Users can have a role in the system such as 'Teacher', 'HelpDesk', 'Administrator', 'Student' etc. Those roles have no influence on the role of a user within a course (except for 'Administrator'). UMRA can specify the role when creating a user, or modify roles of existing users. A user can have only one system-wide role. Like courses, users have 3 userkeys. Those keys are defined as System variables. The userkeys can have multiple values. There is also a 4th user key named usergroup. This key can have only one value and is mostly used to identify the class of a user, although it is not enforced to do so. The usergroup is also defined by a system variable. In umra it is possible to add keys to a user. When the usergroup is specified in the 'TeleTOP User check key' action, the value specified for the key will overwrite any existing value for this key in the specified user. To disable a user use the 'TeleTOP Edit user' action and set the 'Disable' property to 'Yes'. It is advised to add a special key to the user when it is being disabled to easily filter on users disabled by UMRA when users are retrieved from the TeleTOP environment by retrieving the members of a course. Because current limitations in the TeleTOP API, UMRA only retrieves the 'User ID' of the members. Therefore UMRA has to do an extra call per user to retrieve the other details. To retrieve the keys assigned to the user an extra call has to be done per user. This makes

the process slow. However, when TeleTOP is the target environment, user information is mostly not used, just overwritten with new data. Therefor the retrieval of the user and key data is optional to speed up synchronization scripts.

### **User and course keys**

User keys and course keys are cached in a Tools4ever Powershell Agent session. This means, when a user key is used in the session, its internal ID is cached by UMRA. When a user key is removed and recreated with the same name thru the web interface, while the Tools4ever Powershell Agent session is still open, the cache will not know of this manual change and the user or course key will not be properly functioning until the Tools4ever Powershell Agent session is recreated.

Through the web-interface, user and course keys of the same type and with the same name are allowed to be created. However it is not possible to see the difference between those 2 keys when assigning them to a course or user, or when they are used in a filter. Therefor UMRA will not allow the creation of keys of the same type with the same name. When assigning existing keys to a user or course, UMRA will choose the first one it encounters with the given name of the given type.

### **System variables**

System variables can be set through the 'Administration' section in the LMS. Open the item 'System variables' in the 'Administration' section to manage the system variables.

### **Setting up a sync**

#### **Preparations**

In TeleTOP it is currently not possible to retrieve all the users in the system. However it is possible to retrieve all the users from a specific course. Therefore, to setup a successful sync with TeleTOP all users that

should be synchronized, must be member of a course. This can be easily achieved by means of the web interface. Keep in mind, that currently courses can only contain 800 members. So to sync more as 800 users, it is necessary to have multiple courses. Also, make sure the system variable 'Years' contain a value for the current year. (see 'Courses' and 'System variables' for more information.

First create a new course used for synchronization purposes. To create such a course follow the steps bellow:

1. Go to the TeleTOP LMS web interface and choose course management.
2. Click the button 'New'.
3. Specify a name for the course. If multiple courses are required (because of the 800 users limit) add a number to the name (umra\_students\_1, umra\_students\_2... umra\_students\_10 etc...).
4. In the tab 'Authorizations' specify the 'Access to this course'. Select the radio button 'Teachers only' so teachers can modify this course.
5. Uncheck all the checkboxes.
6. Add the users that should be synchronized as 'Guests' to this course. (Do not add more as 800 users to a single course.)
7. Click the button 'Save' Repeat those steps if multiple courses have to be made.

Make sure the Tools4ever Powershell Agent is able to connect to the TeleTOP environment.

Create sync scripts.

### **Ports and connections**

The Tools4ever PowerShell Agent (TPA) must be able to create the following connections.

443 to the TeleTOP server

80 to [www.imsglobal.org](http://www.imsglobal.org)



DNS name resolving of `www.imsglobal.org` must be possible by the powershell agent.

If the hostname of the TeleTOP server is specified as an IP-Address the subject of the certificate must be the same IP-Address else the connection will fail.

If the hostname of the TeleTOP server is specified as an FQDN or as an NetBIOS name the TPA must be able to resolve that name to an IP-Address and the subject of the certificate must be equal to the specified hostname else the connection will fail.

If the certificate is untrusted by the TPA or if the certificate is expired the connection will fail.

If the certificate path is invalid the connection will fail.

### **TeleTOP**

The following procedure is based on the template synchronization project. See *SOAP Synchronization template project* for more information.

In UMRA create a new initialization script with 'TeleTOP' as product. Replace step 5 with the following steps:

1. a. Add the action 'Set variable'
  - b. Set the property 'Variable name' to '%TeleTOPSystem%'
  - c. Set the property 'Value type' to 'text'
  - d. Set the property 'Value' to <the url to LMS>
  - e. Set the On error to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Error, could not set the system name to '<the url to LMS>'.'
2. a. Add the action 'Set variable'

- b. Set the property 'Variable name' to '%TeleTOPSyncCourseCodePre%'
- c. Set the property 'Value type' to 'text'
- d. Set the property 'Value' to <The name of the courses to read the users from for the sync, with out the number. ('umra\_students\_')>
- e. Set the On error to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Error, could not set the main course code to '<the value>'.'

3. a. Add the action 'Set variable'

- b. Set the property 'Variable name' to '%TeleTOPLoginName%'
- c. Set the property 'Value type' to 'text'
- d. Set the property 'Value' to <The login name of the admin user.>
- e. Set the On error to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Error, could not set the login name to '<the login name value>'.'

4. a. Add the action 'Set encrypted variable'

- b. Set the property 'Variable name' to '%TeleTOPPassword%'
- c. Set the property 'Value' to <The password of the admin user.>
- d. Set the On error to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Error, could not encrypt the administrator password.'

5. a. Add the action 'TeleTOP Setup connection'

- b. Set the On error to: A jump to label 'ERROR\_CLEANUP\_POWERSHELL', Set variable '%OperationStatus%' to

'TeleTOP Sync init: Error, could not connect to the TeleTOP system at '%TeleTOPSystem%' as '%TeleTOPLoginName%'.'

Step 8 in the initialization script needs to be replaced with code to retrieve the users from the different courses. Because the number of courses is not known when the script is first run the number in the name will be incremented until the course cannot be found. So the script starts with 'umra\_students\_1' retrieves the members and looks for 'umra\_students\_2' if it does exist it will add the users to the user table and looks for 'umra\_students\_3' else the script stops with looking. etc. This is accomplished with the following steps:

1. a. Add the action 'Set variable'

b. Set the property 'Variable name' to '%TeleTOPSyncCourseCodeNumber%'

c. Set the property 'Value type' to 'numeric'

d. Set the property 'Value' to 0

2. ## An empty table, to which the members of several courses will be appended. This is because we have to query several 800 user courses to retrieve all users. Now they are stored in one table so the join will work as designed.

a. Add the action 'Manage table data'

b. Set the property 'Table data operation' to 'Create table'

c. Set the property 'Table data variable' to '%TeleTOPUsers%'

d. Set the property 'Number of columns' to '29'

3. ## Without this action the table will not have column names and the join in the main sync script will not be able to join this table with the source data.

- a. Add the action 'Manage table data'
- b. Set the property 'Table data operation' to 'Set column name'
- c. Set the property 'Table data variable' to '%TeleTOPUsers%'
- d. Set the property 'Column index' to '0'
- e. Set the property 'Name of column' to 'UserId'

- 5. a. Add the action 'No operation'
- b. Set the label to 'TeleTOP\_loadusers\_start'

- 6. a. Add the action 'Update numeric variable'
- b. Set the property 'Numeric data operation' to 'Increment variable value by 1'
- c. Set the property 'Numeric data variable' to '%TeleTOPSyncCourseCodeNumber%'

- 7. a. Add the action 'Set variable'
- b. Set the property 'Variable name' to '%TeleTOPCourseCode%'
- c. Set the property 'Value type' to 'text'
- d. Set the property 'Value' to '%TeleTOPSyncCourseCodePre%%TeleTOPSyncCourseCodeNumber%'
- e. Set the property 'Resolve immediatly' to 'Yes'

- 8. a. Add the action 'TeleTOP Get course exists'
- b. Set the property 'Course code' to '%TeleTOPCourseCode%'
- c. Set the output property 'Course exists' to '%TeleTOPCourseExists%'

d. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Error, could not determine if course '%TeleTOPCourseCode%' exists.'

9. a. Add the action 'If-Then-Else'

b. Add a if-criteria (Variable Name = '%TeleTOPCourseExists%', Variable type = 'boolean (yes/no, true/false)', Operator = 'equal', Value = 'No')

c. Set the 'Then Goto label' to 'END'

10. a. Add the action 'TeleTOP Get course members'

b. Set the output property 'Members' to '%TeleTOPUsersPart%'

c. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Error, could not retrieve the members of course '%TeleTOPCourseCode%'.'

11. a. Add the action 'Manage table data'

b. Set the property 'Table data operation' to 'Add data of other table'

c. Set the property 'Target table data variable' to '%TeleTOPUsers%'

d. Set the property 'Table data variable to add' to '%TeleTOPUsersPart%'

e. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync init: Could not add the found members in '%TeleTOPUsersPart%' to the user table '%TeleTOPUsers%'.'

12. a. Add the action 'Go to label'

b. Set the property 'Label' to 'TeleTOP\_loadusers\_start'

Save the script.

In UMRA create a new cleanup script with 'TeleTOP' as product and save the script.

In UMRA Create a new Automation script. The purpose of this script is to join the TeleTOP users with the users in the source or destination system. In this example TeleTOP is the destination product. The source is the example source script. The following script will create a table which will contain which users exist in the Source system, which users exist in the TeleTOP system and which users exist in both. For each record in this table a sub-script will be called. The sub-script will be created in the following chapter

1. Create a new Automation Project ('TeletopSync')
2. a. Add the action 'Execute Script'
  - b. Set the property 'Project' to 'SourceSync\_Init' (The name of the Source-Product initialization script)
  - c. Set the On error to: A jump to label 'END'
3. a. Add the action 'Execute Script'
  - b. Set the property 'Project' to 'TeleTOPSync\_Init' (The name of the TeleTOP initialization script)
  - c. Set the On error to: A jump to label 'END'
4. a. Add the action 'Join table data'
  - b. Set the property 'Input table variable 1' to '%ExampleSourceUsers%'

- c. Set the property 'Input table variable 2' to '%TeleTOPUsers%'
- d. Set the property 'Output table variable' to '%JoinedUsers%'
- e. Set the property 'Join condition table 1 column name' to 'EmployeeId'
- f. Set the property 'Join condition table 2 column name' to 'UserId'
- g. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP sync: Error, could not join the example source data '%ExampleSourceUsers%' with the table '%TeleTOPUsers%'.'

5. a. Add the action 'For-Each'

- b. Set the property 'Table variable name' to '%JoinedUsers%'
- c. Set the property 'script project' to 'TeleTOPSync\_Record'
- d. Set the property 'Specify project input variables' to 'Pass 'for each' variables plus: '%G\_TeletopPowerShellSession%, %TeleTOPSyncCourseCodePre%'
- e. Specify the following column mapping: (The columns defined in the source table, this may vary, depending on the source data, followed by the 29 columns of the TeleTOP user table, followed by the join result variable.)

Column\_01 -> %EmployeeId%

Column\_02 -> %FirstName%

Column\_03 -> %MiddleName%

Column\_04 -> %LastName%

Column\_05 -> %PartnerFirstName%

Column\_06 -> %PartnerMiddleName%

Column\_07 -> %PartnerLastName%

Column\_08 -> %NamingConvention%

Column\_09 -> %Title%

Column\_10 -> %Gender%

Column\_11 -> %Birthday%

Column\_12 -> %ClassName%

Column\_13 -> %StartDate%

Column\_14 -> %EndDate%

Column\_15 -> %HomePhone%

Column\_16 -> %MobilePhone%

Column\_17 -> %TeleTOPUserId%

Column\_18 -> %TeleTOPAccountId%

Column\_19 -> %TeleTOPRoleInCourse%

Column\_21 -> %TeleTOPUserType%

Column\_22 -> %TeleTOPFirstName%

Column\_23 -> %TeleTOPMiddlename%

Column\_24 -> %TeleTOPLastName%

Column\_25 -> %TeleTOPInitials%

Column\_26 -> %TeleTOPFullName%

Column\_27 -> %TeleTOPGender%

Column\_28 -> %TeleTOPStreet%

Column\_29 -> %TeleTOPHouseNumber%

Column\_30 -> %TeleTOPCity%

Column\_31 -> %TeleTOPPostalCode%

Column\_32 -> %TeleTOPCountry%

Column\_33 -> %TeleTOPEmail%



Column\_34 -> %TeleTOPPhoneNumber%

Column\_35 -> %TeleTOPCellularNumber%

Column\_36 -> %TeleTOPFaxNumber%

Column\_37 -> %TeleTOPWebsite%

Column\_38 -> %TeleTOPCompanyInstitute%

Column\_39 -> %TeleTOPFunction%

Column\_40 -> %TeleTOPAboutMe%

Column\_41 -> %TeleTOPUserRole%

Column\_42 -> %TeleTOPBlocked%

Column\_43 -> %TeleTOPUserKey1%

Column\_44 -> %TeleTOPUserKey2%

Column\_45 -> %TeleTOPUserKey3%

Column\_46 -> %TeleTOPUserGroup%

Column\_47 -> %JoinResult%

f. Set the On error to: A jump to label 'END'

6. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

7. a. Add the action 'No operation'

b. Set the label to 'ERROR'

8. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

9. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'
10. a. Add the action 'No operation'
  - b. Set the label to 'END'
11. a. Add the action 'Execute Script'
  - b. Set the property 'Project' to 'TeletopSync\_Cleanup' (The name of the TeleTOP cleanup script.)
12. Specify a scheduling scheme in the schedule tab.

Creating the TeleTOP sub script.

This script will be run for every record in the 'join' table. The join table consists of all the data of a user in the source and/or destination system plus a column providing information about the system in which the user exists. For every record in the join-table the new script will be called. The data in the record is saved into variables defined in the For-Each action in the previous chapter. The %JoinResult% variable stores in which system the user exists. The user is matched by its UserId in TeleTOP and its EmployeeId in the source data. This is specified in the Join action in the previous script.

The %JoinResult% variable can contain the following values:

- 0 - The user exists in both systems
- 1 - The user exists in the 'Input table variable 1' of the Join table action. (In this example the user is found in the source system, but must be created in TeleTOP)

2 - The user exists in the 'Input table variable 2' of the Join table action. (In this example the user is found in TeleTOP, but does not exist in the source and will therefore be disabled in TeleTOP)

any other value means an error occurred. This is mostly caused by an incorrect number of column specifications in the For-Each.

To create a TeleTOP sub-script follow the next steps.

1. Create a new Automation project 'TeleTOPSync\_Record'.

2. a. Add the action 'Map variable'

b. Set the property 'Input variable' to '%JoinResult%'

c. Set the property 'Output variable' to '%Operation%'

d. Set the property 'Table entry 1' to '0' -> 'equal'

e. Set the property 'Table entry 2' to '1' -> 'teletop\_create'

f. Set the property 'Table entry 2' to '2' -> 'teletop\_disable'

g. Set the property 'Default value of output variable' to 'ERROR\_INVALID\_JOINVALUE'

3. a. Add the action 'Go to label'

b. Set the property 'Label' to '%Operation%'

4. a. Add the action 'No operation'

b. Set the label to 'equal'

5. a. Add the action 'Execute Script'

b. Set the property 'Project' to 'TeleTOPSync\_UpdateUser'

c. Set the On error to: A jump to label 'END'

6. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

7. a. Add the action 'No operation'

b. Set the label to 'teletop\_create'

8. a. Add the action 'Execute Script'

b. Set the property 'Project' to 'TeleTOPSync\_CreateUser'

c. Set the On error to: A jump to label 'END'

9. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

10. a. Add the action 'No operation'

b. Set the label to 'teletop\_disable'

11. a. Add the action 'Execute Script'

b. Set the property 'Project' to 'TeleTOPSync\_DisableUser'

c. Set the On error to: A jump to label 'END'

12. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

13. a. Add the action 'No operation'

b. Set the label to 'ERROR\_INVALID\_JOINRESULT'

14. a. Add the action 'Get file/directory info'

b. Set the property 'Target file/directory' to '\$: ERROR INVALID PATH \\ \$:'

c. Set the property 'Error if not found' to 'Yes'

d. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP sync record: Error, the join result '%JoinResult%' is invalid.'

15. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

16. a. Add the action 'No operation'

b. Set the label to 'ERROR'

17. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

18. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

19. a. Add the action 'No operation'

- b. Set the label to 'END'

Save the project

TeleTOP Sync update user.

To update an existing user, the TeleTOP user must be edited. In this topic a very simple edit user script is specified. Of course in real/life situations a name generation script should be called before edit the user, to utilize the partner name information in the source data, or any other client wishes to convert the source name information to the wished name convention in TeleTOP. The same is for the e-mail address. This step is skipped here.

Create the script by following the next steps:

1. Create a new Automation project 'TeleTOPSync\_UpdateUser'.

2. a. Add the action 'TeleTOP Edit user'

- b. Set the property 'Password' to <do not specify a value for this property>

- d. Set the property 'Initials' to <do not specify a value for this property>

- d. Set the property 'Gender' to '%Gender%'

- e. Set the property 'E-Mail address' to '%EmployeeId%@organisation.com'

- f. Set the property 'Telephone number' to '%HomePhone%'

- g. Set the property 'Cellular number' to '%MobilePhone%'

- h. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP sync update user: Error, could not update user '%EmployeeId%'.'

3. a. Add the action 'Execute Script'
  - b. Set the property 'Project' to 'TeleTOPSync\_SetUserClass'
  - c. Set the On error to: A jump to label 'END'
4. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'
5. a. Add the action 'No operation'
  - b. Set the label to 'ERROR'
6. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>
7. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'
8. a. Add the action 'No operation'
  - b. Set the label to 'END'

TeleTOP Sync Create user.

To create a new user in the TeleTOP environment, all information about the user should be available. Because of the variables utilized by the join, this information is available. Of course in real/life situations a name

generation script should be called to convert the source data to the wished destination format. For example, to utilize the partner name information in the source data. This step is skipped in this example.

Create the script by following the next steps:

1. Create a new Automation project 'TeleTOPSync\_CreateUser'.
2. Add the action 'Generate password'
3. a. Add the action 'TeleTOP Create user'
  - b. Set the property 'Full name' to <do not specify a value for this property>
  - c. Set the property 'Initials' to <do not specify a value for this property>
  - d. Set the property 'FullName' to '%FirstName% %MiddleName% %LastName%'
  - e. Set the property 'Gender' to '%Gender%'
  - f. Set the property 'E-Mail address' to '%EmployeeId%@organisation.nl'
  - g. Set the property 'Telephone number' to '%HomePhone%'
  - h. Set the property 'Cellular number' to '%MobilePhone%'
  - i. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP sync create user: Error, could not create the user '%EmployeeId%'.'
4. a. Add the action 'Execute Script'
  - b. Set the property 'Project' to 'TeleTOPSync\_SetLastStudentCourse'



c. Set the On error to: A jump to label 'END'

5. a. Add the action 'Execute Script'

b. Set the property 'Project' to 'TeleTOPSync\_SetUserClass'

c. Set the On error to: A jump to label 'END'

6. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

7. a. Add the action 'No operation'

b. Set the label to 'ERROR'

8. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

9. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

10. a. Add the action 'No operation'

b. Set the label to 'END'

TeleTOP Sync disable user.

To disable an existing user in the TeleTOP environment. The user must only be blocked. However to ease the search for disabled user thru the web interface, the users will be added to the 'umra\_disabled' user

group. (In this example the user group represent the class.) See the 'TeleTOPSync\_SetUserClass' script creator in the next topic.

Create the disable script by following the next steps.

1. Create a new Automation project 'TeleTOPSync\_DisableUser'.

\* WARNING THIS SCRIPT WILL DISABLE EVERY USER WHICH IS NOT FOUND IN THE SOURCE DATABASE, IMPLEMENT AT YOUR OWN RISK.

\* Remove step 2 to 6 to just log when this script is called without disabling the user.

2. a. Add the action 'TeleTOP Edit user'

b. Set the property of all properties to <do not specify a value for this property>

c. Set the property 'User ID' to '%TeleTOPUserId%'

d. Set the property 'Disable' to 'Yes'

e. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP sync disable user: Error, could not disable the user '%EmployeeId%'.'

3. a. Add the action 'Set variable'

b. Set the property 'Variable name' to '%ClassName%'

c. Set the property 'Value type' to 'text'

d. Set the property 'Value' to 'umra\_disabled'

4. a. Add the action 'Execute Script'

b. Set the property 'Project' to 'TeleTOPSync\_SetUserClass'

c. Set the On error to: A jump to label 'END'

5. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

6. a. Add the action 'No operation'

b. Set the label to 'ERROR'

7. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

8. a. Add the action 'Go to label'

b. Set the property 'Label' to 'END'

9. a. Add the action 'No operation'

b. Set the label to 'END'

#### TeleTOP Sync set user class

To specify the class the TeleTOP user key 'usergroup' will be used. First UMRA has to check if the user key exists. If it does not exist, it has to be created, before it can be assigned to a user. The following script will perform those tasks.

1. Create a new Automation project 'TeleTOPSync\_SetUserClass'.

2. a. Add the action 'TeleTOP Get key exists'
  - b. Set the property 'Key Name' to '%ClassName%'
  - c. Set the property 'Key Type' to 'usergroup'
  - d. Set the output property 'Key Exist' to '%TeleTOPClassExists%'
  - e. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set user class: Error, could not check if the user key '%ClassName%' of the type 'usergroup' exists.'
3. a. Add the action 'If-Then-Else'
  - b. Add a if-criteria (Variable Name = '%TeleTOPClassExists%', Variable type = 'boolean (yes/no, true/false)', Operator = 'equal', Value = 'No')
  - c. Set the 'Then Goto label' to 'teletop\_class\_create'
  - d. Set the 'Else Goto label' to 'teletop\_class\_exist'
4. a. Add the action 'No operation'
  - b. Set the label to 'teletop\_class\_create'
5. a. Add the action 'TeleTOP Create key'
  - b. Set the property 'Key Name' to '%ClassName%'
  - c. Set the property 'Key Type' to 'usergroup'
  - d. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set user class: Error, could not create the user key '%ClassName%' of the type 'usergroup' although it does not exist.'
6. a. Add the action 'No operation'
  - b. Set the label to 'teletop\_class\_exist'

7. a. Add the action 'TeleTOP User check key'
  - b. Set the property 'User ID' to '%EmployeeId%'
  - c. Set the property 'Key Name' to '%ClassName%'
  - d. Set the property 'Key Type' to 'usergroup'
  - e. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set user class: Error, could not add the user key '%ClassName%' of the type 'usergroup' to the user '%EmployeeID%'. '
8. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'
9. a. Add the action 'No operation'
  - b. Set the label to 'ERROR'
10. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>
11. a. Add the action 'Go to label'
  - b. Set the property 'Label' to 'END'
12. a. Add the action 'No operation'
  - b. Set the label to 'END'

TeleTOP Sync set last student course.

1. a. Add the action 'Set variable'
  - b. Set the property 'Variable name' to '%TeleTOPSyncCourseCodeNumber%'
  - c. Set the property 'Value type' to 'numeric'
  - d. Set the property 'Value' to 0
5. a. Add the action 'No operation'
  - b. Set the label to 'TeleTOP\_studentcourse\_search'
6. a. Add the action 'Update numeric variable'
  - b. Set the property 'Numeric data operation' to 'Increment variable value by 1'
  - c. Set the property 'Numeric data variable' to '%TeleTOPSyncCourseCodeNumber%'
7. a. Add the action 'Set variable'
  - b. Set the property 'Variable name' to '%TeleTOPCourseCode%'
  - c. Set the property 'Value type' to 'text'
  - d. Set the property 'Value' to '%TeleTOPSyncCourseCodePre%%TeleTOPSyncCourseCodeNumber%'
  - e. Set the property 'Resolve immediatly' to 'Yes'
8. a. Add the action 'TeleTOP Get course exists'

- b. Set the property 'Course code' to '%TeleTOPCourseCode%'
- c. Set the output property 'Course exists' to '%TeleTOPCourseExists%'
- d. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set last student course: Error, could not determine if course '%TeleTOPCourseCode%' exists.'

9. a. Add the action 'If-Then-Else'

- b. Add a if-criteria (Variable Name = '%TeleTOPCourseExists%', Variable type = 'boolean (yes/no, true/false)', Operator = 'equal', Value = 'No')
- c. Set the 'Then Goto label' to 'TeleTOP\_studentcourse\_create'

10. a. Add the action 'TeleTOP Get course members'

- b. Set the output property 'Members' to '%TeleTOPUsersPart%'
- c. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set last student course: Error, could not retrieve the members of course '%TeleTOPCourseCode%'.'

11. a. Add the action 'Manage table data'

- b. Set the property 'Table data operation' to 'Determine number of rows'
- c. Set the property 'Table data variable' to '%TeleTOPUsersPart%'
- d. Set the property 'Number of rows returned in variable' to '%TeleTOPUserCount%'
- e. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set last student course: Could not count the members of course '%TeleTOPCourseCode%' in '%TeleTOPUsersPart%'.'

12. a. Add the action 'If-Then-Else'

b. Add a if-criteria (Variable Name = '%TeleTOPUserCount%', Variable type = 'numeric', Operator = 'greater than', Value = '750')

c. Set the 'Then Goto label' to 'TeleTOP\_studentcourse\_search'

d. Set the 'Else Goto label' to 'TeleTOP\_studentcourse\_set'

13. a. Add the action 'No operation'

b. Set the label to 'TeleTOP\_studentcourse\_create'

14. a. Add the action 'TeleTOP Create course'

b. Set the property 'Course Name' to 'UMRA: %TeleTOPCourseCode%'

d. Set the property 'Course Description' to 'DO NOT DELETE, THIS COURSE IS MAINTAINED BY UMRA'

d. Set the property 'TeleTOPSourceCourseCode' to '%TeleTOPSyncCourseCodePre%1'

e. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set last student course: Error, could not create a new course '%TeleTOPCourseCode%' based on '%TeleTOPSyncCourseCodePre%1'.'

15. a. Add the action 'No operation'

b. Set the label to 'TeleTOP\_studentcourse\_set'

16. a. Add the action 'TeleTOP Add member to course'

b. Set the property 'Role type' to 'Member'

c. Set the On error to: A jump to label 'ERROR', Set variable '%OperationStatus%' to 'TeleTOP Sync set last student course: Error,



could not add the user '%EmployeeId%' to the course  
'%TeleTOPCourseCode%'.'

17. a. Add the action 'Go to label'
- b. Set the property 'Label' to 'END'

18. a. Add the action 'No operation'
- b. Set the label to 'ERROR'

19. <place some action to process the '%OperationStatus%' variable (for example the action 'Log specific variables' or 'Export variables' or call a generic log script with the 'Execute script' action>

20. a. Add the action 'Go to label'
- b. Set the property 'Label' to 'END'

21. a. Add the action 'No operation'
- b. Set the label to 'END'

## **It's Learning**

### **Description**

"It's Learning" is a web based Learning Management System hosted by a company of the same name. See <http://www.itslearning.co.uk> for more information. UMRA projects can interface to this system in order to perform specific tasks directly related to account and group management. UMRA Interfaces with It's learning as specified in the "IMS Enterprise Services" specification (<http://www.imsglobal.org/es/index.html>).

This is completely transparent in UMRA. The underlying functionality is implemented in the Powershell agent of UMRA. Within UMRA we have developed dedicated It's Learning script actions that can be directly used in an UMRA script, in order to create and manage persons, groups of various kinds, and memberships.

Note that the UMRA actions are not designed to create any specific learning content.

### **Prerequisites**

- 1) The UMRA Powershell Agent service must be installed. For installation instructions see the chapter about the Powershell agent service in the UMRA user guide.
- 2) A working It's learning system, accessible by browser from the computer running the powershell service.

### **Ports and connections**

The Tools4ever PowerShell Agent (TPA) must be able to create the following connections:

443 to the It's Learning server

80 to [www.imsglobal.org](http://www.imsglobal.org)

DNS name resolving of [www.imsglobal.org](http://www.imsglobal.org) must be possible by the powershell agent.

### **Setting up a connection to an It's learning system, and executing It's learning actions**

- 1) Set up a dedicated Powershell session

All communication with It's learning is done by means of Powershell commands/scripts that are automatically generated by the UMRA script actions, and executed by the UMRA Powershell service. The Powershell service is not only used by the It's learning script actions, but also by many other actions. To prevent interference with other unrelated

actions it is recommended to run all It's learning actions in a dedicated Powershell session.

This is done with *Script Action: Setup Powershell Agent service session*. This creates a new Powershell session, and returns the ID of this newly created session. This session ID should then be specified as input for all related It's learning actions.

## 2) Open the connection it's learning with **Script Action: It's learning Setup Connection**

As input specify the just created powershell session ID , and the login credentials for the It's learning system. Note that the specified password must be encrypted. So use for example *Script Action: Set encrypted variable* on page 546 to encrypt the password. Note also that the account used for the login must be an administrative user with "web service rights" in It's learning. This may be a different account than you use to access the normal It's Learning web interface.

3) Next, you can use all It's Learning actions, using the Powershell session ID for the It's Learning connection parameter.

## Closing connections

When finished using the actions in the script use the action **It's learning Close Connection** to close the connection to It's learning system, and lastly *Script Action: Release Powershell Agent service session* on page 606 to release the Powershell session used.

## Note on Powershell sessions

Setting up a new powershell session and It's learning connection is a relative resource intensive operation, so its recommended to use the same Powershell session for all It's Learning actions when possible, instead of frequently setting up and releasing sessions.

A Powershell Session may be automatically released by the Powershell service if it has nothing to do for a longer period. In order to check if a certain earlier created powershell session used earlier is still available, use *Script Action: Check Powershell Agent service session*.

### **It's Learning concepts**

Basically, there are two kinds of objects in the It's learning environment that can be managed with UMRA, i.e **Persons And Groups**.

Groups are created in a hierarchy. (that means they are located in a tree structure, they have a 'parent' object, usually another group). Persons are not created in a hierarchy. (that means they are a flat table). Persons can be members of several groups. With UMRA you can create, delete, and edit persons and groups, and assign and list memberships.

Groups come in different flavors. Examples of these are "Site", "School", "Study", "Studyyear", "class", "course". These are all a kind of group, but there are special actions in UMRA to create them more easily.

#### 4.3.11. Variable actions

##### Table

*Script Action: Generate generic table*

##### Function

Script action to generate a generic table (resulting from a LDAP query, database query etc, csv files, etc) and store the entire results in a special "table" variable for further processing.

##### Deployment

This script action is used when you need to create a table with query results in a script. The variable which is the output of this script action can be further processed using any other script action that accept table(s) as input. Most prominently this is the *Script Action: Manage table data* on page 528. Also, the resulting table can be shown in a form, when the script is part of a form project, and run before the form is displayed.

##### Properties

The Properties of this action are configured by means of special configuration dialogs.

Press the configure button in the properties tab on the properties page of this action to access the dialog, and press F1 for the online help of the specific dialog for more information.

##### Remarks

##### Script Tables and Form tables

There are two distinct tables types in UMRA. **Script Tables** and **Form tables**

A **Script table** is a table contained in a script table variable, for example generated with the script action Generate generic table. Subsequent script actions that accept a table variable as input may act on this table to perform a wide range of actions. It can be used in any UMRA module that uses a script.

A **Form table** is a special **Form field** within a UMRA Form. It is used to display the contents of a table in a form. Obviously form table only exist

in the "Forms and delegation" module. The required queries for this form table are contained in the definition of the form field itself, and executed when the form is presented to the user.

Both the **Script table** and the **Form table** can be the result of queries to for example LDAP directories, csv files, and databases, and these query definitions have common configuration interfaces and capabilities.

Alternatively a **Form table** element can build itself from a pre-existing **Script table** variable. This is used for instance if it is required that the table result should be modified before being presented the form. In that case the build-in query functionality in the **Form table** cannot be used.

#### **64 bit Operating Systems.**

On 64 bit Operating systems, UMRA runs as a normal 32 bit application. This means that in order to communicate with providers for database connections (e.g. ODBC), It is required that the appropriate 32 bit odbc data sources are installed c.q. configured. Make sure to start the odbcad32.exe program located in the SysWOW64 folder, and not the program with the same name in the system32 folder.

#### **See also:**

*Script Action: Manage table data on page 528*

*Script Action: Manage table data*

For most standard User Management tasks, the available built-in tables (Fixed, Network, Generic) can be used. There are situations however, where these standard tables may not be sufficient:

- You may wish to evaluate an existing table programmatically and / or create a new table from scratch ;
- You may have to combine data from one of the built-in tables with data which are not contained in a table.

In such cases, you can use the **Manage table data** script action to create your own tables. This script action includes many operations for creating and editing tables:

Table data operation	Description
Create table	Creates an empty table. Initially, the table contains no rows. The number of columns must be specified.
Append a row at the end of the table	Adds a row at the end of a table. Initially, the table row contains empty data text fields. The specified row index variable will contain the index of the new row after the action has executed.
Append a column at the end of the table	Adds a column at the end of a table. Initially, the table column contains empty data text fields. The specified column index variable will contain the index of the new column after the action has executed.
Set the data for the specified row and column of the table	Sets the data value of the specified cell to the specified value. The cell is specified by the row and column index. The data cell value can be specified by a text value or variable name.
Get the data at the specified row and column of the table	Copies the data value of the specified cell into the value of the specified output variable. The cell is specified by the row and column index.
Get the number of table rows	Determines the number of rows in a table. The number is stored in the row count variable.
Get the number of table columns	Determine the number of table columns. The number is stored in the specified variable.
Copy row	Copy a specific row to another table. The row is specified by its index. The target table should either not exist or have the same number of columns. If the target table does not exist, it is created. If the target table does exist, the new row is added at the end of the table. If the target table does exist and already contains data, but if the table does not contain the same number of columns, an error is generated.

Copy multiple rows	Copy multiple rows to another table. The rows are specified by their indexes. The indexes are stored in another table with only a single column and one row for each index. The target table should either not exist or have the same number of columns of the original table. If the target table does not exist, it is created. If the target table does exist, the new rows are added at the end of the table. If the target table does exist and already contains data, but if the table does not contain the same number of columns, an error is generated.
Copy table	Copy one table to another table. If the destination table variable already exists, its data is overwritten.
Remove a specified row	Removes the row with a specified row number from the table.
Remove multiple rows	Remove multiple rows from a table. The rows are specified by their indexes. The indexes are stored in another table with only a single column and one row for each index.
Remove duplicate rows	Removes duplicate rows from a table. The specified key column index is used to find duplicate rows. A duplicate is found when the cell data of two rows in the specified key column are equal.
Remove a specified column	Removes the column with a specified column number from the table
Sort on column index	Sort the rows of the table based on the specified column. The column is specified using the index number of the column (0,1,...,[number of columns]-1).
Sort on column name	Sort the rows of the table based on the specified column. The column is specified using the name of the column.
Convert multi-text variable to table	Converts a multi-text variable into a single column of a table.



Convert table column to multi-text variable	Converts the contents of the specified column of the table to a multi-text variable
Convert multi-value variable to table	Converts a multi-value variable into a single column of a table
Set column name	Set the name of the specified column. The column is specified by its index.
Replace column name	Replace the name of the specified column by the new name of the column. The column is specified by its original name.
Get column name	Get the name of the specified column. The column is specified by its index.
Complete rows	Complete all rows of the specified table. If one or more rows have less columns compared to other rows, empty text values are appended to these rows to complete them.

Search table	<p>Search all rows in the table. Either a specific column or all columns of each row are searched for. To search only in a specific column, specify the column index in the field 'Optional: Search in column'. Result indexes are stored in the 'Result output table'. The search can be performed in various ways:</p> <p>1: Search for a specific text Specify the text as 'Search text'. By default, a match is found if the text in the table contains the specified search text. To require an exact match (e.g. match only if the contents of the table cell equals the specified text), include the keyword STRICT in the field 'Optional: Search column'. Examples: 0, STRICT or STRICT. The search is always case-insensitive.</p> <p>2: Search for empty cells Leave the 'Search text' field empty. A match is found if the table cell contains no data or an empty text value.</p> <p>3: Search for not-empty cells Specify [*] for the 'Search text' field. A match is found if the table cell contains data.</p> <p>The result is stored in the specified 'Result output table'. For each match, a row is created in the output table. The first column holds the matching row number (0,1,2,...). The second column the matching column number (0,1,2...).</p>
Add data of table variable to table	Appends the contents of an other table to the current table. both tables must contain the same number of columns and data-types.
Log table data	Writes the contents of a table to the log file

Export to .csv file	<p>Exports the table data to a .CSV file. You need to specify the file name, the separator and format string (optional). The format string can contain the following keywords:</p> <p>UNICODE:           File will be saved in UNICODE format.</p> <p>APPEND:            If the file exists, the data is appended to the existing file.</p> <p>SINGLEQUOTE:    The quote character used is ' . By default the quote character is " .</p> <p>QUOTEBLANK:    Quotes will be added if a text field contains a space or tab.</p> <p>QUOTEALWAYS: Quotes will be placed around all text fields.</p> <p>HEADER:           The first line of the output will contain the column name</p> <p>Specify multiple keywords using the comma's.</p>
---------------------	---

*Script Action: Join table data*

### Function

Joins the table data from table data variable A and table data variable B and returns the result in output table data variable C. This output table includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables.

In database query terminology this is called a full outer join. Full outer joins return results which include rows from table A and B, whether or not they contain matching values in the joining columns. In the result set, the columns for which no match was found will contain empty text fields.

### Deployment

This script action is typically used for identity management tasks in heterogeneous network environments. For instance, a company may have a need to update Active Directory with the latest phone information from a phone system database. Using the **Join table data** script action, the table data from both systems can be joined. The leading information from the phone system can then be matched

against the information found in Active Directory. Each row of the joined table is evaluated. If the User IDs match and the phone number in Active Directory differs from the phone number found in the phone system, Active Directory will be updated with the phone number from the phone system.

**Example: Synchronization procedure**

Consider two sets of table data, both generated in UMRA. The first table contains the result of a query on an SQL database holding the company's telephone numbers. The table contains a column with user IDs and a column with phone numbers. Table B is the result of an LDAP query on Active Directory and contains columns with the employee ID (**Employee ID** column), telephone number (**Telephone** column) and the user's distinguished name (**DN** column) from Active Directory.

By joining these two tables, each row of table A will be paired with rows from table B. For a join to succeed, a column of both table A and table B needs to be specified on which to base the join. These columns have to contain matching (non-duplicate) data. In the example shown below, the columns **UserID** and **Employee ID** contain the data on which the join is based.

TABLE A

UserID	Phone
900001	1948904
900002	1948905
900003	1948908
900004	1948900
900005	1948911
900006	1948912

TABLE B

Employee ID	Telephone	DN
900001	1948904	CN=John Smith,CN=Users,DC=Tool
900002	1948913	CN=Paul Anderson,CN=Users,DC=T
900003	1948908	CN=Linda Ewing,CN=Users,DC=Too
900004	1949000	CN=Tracy Jones,CN=Users,DC=Too
900005	1948911	CN=Graham Bell,CN=Users,DC=Too
900006	1948912	CN=Jeffrey Timber,CN=Users,DC=To



Join Table Data

TABLE C

UserID	Phone	Employee ID	Telephone	DN
900001	1948904	900001	1948904	CN=John Smith,CN=Users,DC=Tool
900002	1948905	900002	1948913	CN=Paul Anderson,CN=Users,DC=T
900003	1948908	900003	1948908	CN=Linda Ewing,CN=Users,DC=Too
900004	1948900	900004	1949000	CN=Tracy Jones,CN=Users,DC=Too
900005	1948911	900005	1948911	CN=Graham Bell,CN=Users,DC=Too
900006	1948912	900006	1948912	CN=Jeffrey Timber,CN=Users,DC=To

For each row of table C, a project script is executed using the relevant variables as input for the script. These variables represent the information in the various table columns (if you are not familiar with the use of variables, please read the *UMRA Basics* on page 3 manual first).

This script checks if the specified columns do indeed match (in the example above, it concerns the columns **UserID** and **Employee ID**). If this is the case, the phone numbers in the columns **Phone** and **Telephone** are compared. If these phone numbers are NOT equal (as in the case of the phone numbers for UserID 900002 and 900004 in the example shown above), the value in the **Phone** column with the leading phone information will be used to set the Telephone attribute of the user specified in the **DN** column.

The **Get User on page 31** script action can use the value specified in the **DN** column to retrieve the user object of the specified user in Active Directory. Next, the value for the **telephoneNumber** attribute can be set using the **Set attribute on page 124** script action. This script action can set the value of the **telephoneNumber** attribute to the value found in the **Phone** column for the corresponding user ID.

This is only a simple example of synchronization. More complex synchronization scripts can be created to build a solution which meets the specific needs of your organization (e.g. synchronizing user management data across various directory services supporting LDAP, such as Novell e-Directory and Linux OpenLDAP).

#### **Table variables**

In this section, the table input and output variables for the join operation are specified.

##### **Input table variable 1**

First input table for the join operation.

##### **Input table variable 2**

Second input table for the join operation.

##### **Output table variable**

This variable will contain the table data resulting from the join operation.

##### **Join condition**

**Column with name N of input table 1 must match column with name N of input table 2**

Here you need to specify the name of the columns on which the join is based. The content of the specified columns must match (non duplicates). The name of the column is not necessarily the same.

In case of the example given earlier, the specification would have to be as follows:

**Properties**

Table join

Specify the table join operation

Table variables

Input table variable 1: %TableA%

Input table variable 2: %TableB%

Output table variable: %TableC%

Join condition

Column with name UserID of input table 1 must match column with name Employee ID of input table 2.

Options

☐ Text mode

OK Cancel Apply Help

The content of **Table A** is stored in %TableA% and the content of **Table B** in %TableB%. The join operation is based on column **UserID** of **Table A** and the column **Employee ID** of **Table B**.

#### Log messages

When the **Join table data** script action is used, the log message also reports on the results of the join operation:

Total number of rows: N, common: N, left: N, right: N

**Total number of rows** - total number of rows in the table containing the joined data.

**Common** - the number of rows containing common data.

**Left** - number of unmatched rows from the first (left) table, joined with a NULL row of the second (right) table.

**Right** - number of unmatched rows from the second table (right), joined with the NULL row of the first (left) table.

### Options

**Text mode** - Activate this checkbox to run the join operation in text mode. Numeric data will then be treated as text. Note that if your data are imported from a text file (CSV), these data will be automatically treated as text.

### See also:

*UMRA Basics* on page 3

*Get User script action* on page 31

*Set attribute script action* on page 124

### Database

## CHAPTER

*Script action: Update database*

Script Action: Update database - Database

Click the **Configure** button to configure the connection to the database you wish to update. The connection details will be shown in the **Database specification** window.

### Next steps:

1. *Entering the SQL statement* on page 539 for updating the record set
2. *Defining test variables* (see "Variable list" on page 778) (optional, this is for test purposes only)
3. *Running a test* (see "Script Action: Update database - Test" on page 541)



#### Script Action: Update database - Introduction

This action allows you to connect to any existing database and update the record set. The update is usually the result of a *Manage table data* on page 528 action.

There are several steps involved in updating an existing database:

1. *Specifying the database* (see "Script Action: Update database - Database" on page 538)
2. *Entering the SQL statement* on page 539 for updating the record set
3. *Defining test variables* (see "Variable list" on page 778) (optional, this is for test purposes only)
4. *Running a test* (see "Script Action: Update database - Test" on page 541)

The resulting database update configuration details will be displayed in the Database update action description window. Clicking the **Configure** button will take you to the **Database update commands configuration** window. From here you can specify the database you wish to update and complete the steps as indicated above.

#### Script Action: Update database - SQL Statements

##### **Previous steps:**

##### 1. Specifying the database

In the **SQL statement** window you can specify the SQL syntax for updating the record set for your table. In this section you will find a few examples for update operations in an existing MS Access database. For more detailed information on updating your database, see the instructions of your database provider.

##### **Single record append query in MS Access**

```
INSERT INTO target [(field1[, field2[, ...]])]
```

```
VALUES (value1[, value2[, ...]])
```

##### **Multiple record append query in MS Access**

```
INSERT INTO target [(field1[, field2[, ...]])] [IN externaldatabase]
```

SELECT [*source.*]*field1*[, *field2*[, ...]]

FROM *tableexpression*

The INSERT INTO statement contains the following parts:

Part	Description
<i>target</i>	The name of the table or query to append records to.
<i>field1, field2</i>	Names of the fields to append data to, if following a target argument, or the names of fields to obtain data from, if following a source argument.
<i>externaldatabase</i>	The path to an external database
<i>source</i>	The name of the table or query to copy records from.
<i>tableexpression</i>	The name of the table or tables from which records are inserted. This argument can be a single table name or a compound resulting from an INNER JOIN , LEFT JOIN , or RIGHT JOIN operation or a saved query.
<i>value1, value2</i>	The values to insert into the specific fields of the new record. Each value is inserted into the field that corresponds to the value's position in the list: value1 is inserted into field1 of the new record, value2 into field2, and so on. You must separate values with a comma, and enclose text fields in quotation marks ( ' ').

An example of an SQL statement for appending a single record may look as follows:



The variables %EmployeeID%, %FullName%, %Department%, %Manager%, %Location%, %FirstName% and %Phone% in this example will be replaced with their actual values at runtime.

#### **Do not show statements in log files**

The database statements can contain sensitive information, for instance passwords. To prevent these from being shown in log files, select the option **Do not show statements in log files**. When this option is specified, the statement description is shown, but at runtime, but the actual statement is not shown in the log file. Note that if the statement execution fails, the statement is not shown in the log file in this case if this option is specified.

#### **Next steps:**

3. *Defining test variables* (see "Variable list" on page 778) (optional, this is for test purposes only)
4. *Running a test* (see "Script Action: Update database - Test" on page 541)

Script Action: Update database - Test

#### **Previous steps:**

1. *Specifying the database* (see "Script Action: Update database - Database" on page 538)
2. *Entering the SQL statement* on page 539 for updating the record set
3. *Defining test variables* (see "Variable list" on page 778) (optional, this is for test purposes only)

To validate the SQL statement for the database update, click the **Test** button. Please note that this is not a simulation. The database will be updated according to the SQL statement and the test variables you have provided. The test result will appear in the **Test results** window.



### **Name generation**

*Script Action: Generate name(s)*

#### **Function**

Generates one or more names based on the value of one or more input variables. The algorithm that is used to generate the output names is configurable. The output value names are stored in variables.

#### **Deployment**

This action is typically used in UMRA form projects to propose the user name and full name of a new user account. These names are generated by the algorithm of the script action when the end-user specifies the input names (typically first, middle and last name). The resulting names are presented in a form and the end-user can then accept the names or let the algorithm generate new names (next iteration cycle).

The script actions that create user accounts, *Script Action: Create User (AD)* on page 3 and *Script Action: Create User (no AD)* on page 68 contain a user name generation algorithm. If the generated names are used as input names for these actions, the user name generation algorithm of these actions should not be used. So, when creating user accounts, there are 2 methods to generate user names automatically:

1. Use script action **Generate name(s)** and disable the name generation algorithm of the **Create user** action.
2. Do not use script action **Generate name(s)** and use the name generation algorithm of the **Create user** action instead.

With the first method, the script must implement a loop:

1. Generate name.
2. Check if name is unique.

3. If name is unique, continue with step 4, if not unique go to step 1 to generate next name.
4. Create user account.

This method requires a more complex script. On the other hand, the names that are generated by the algorithm can be shown to the end-user before the account is created.

To configure the name generation action, the following dialog is used:



Use the **Edit** and browse (...) button to edit the currently configured algorithm or import another algorithm.

#### **Iteration - Use internal data (can only be used for mass projects)**

If the name generation action is called multiple times in mass projects, new names are generated according to the configuration of the name generation algorithm. If this option is selected, the iteration mechanism is controlled by the action itself. This option can only be used in mass projects. In form projects, the same names will be generated if this option is selected.

#### **Iteration - Use variable**

If this option is selected, the iteration mechanism is controlled by a numeric variable specified in this field. This variable holds a number that corresponds with the iteration cycle. The first time, the variable is 0. The action will generate the names according to the first iteration cycle.

**Next, the value of the variable is incremented.** This, the next time the action is called during the same session, the variable has a value of 1. Hence, the next iteration cycle of the name generation algorithm is used to generate the names. This process continues.

**Note: The action will create the iteration variable if it does not exist when executed.**

**Properties**

Property Name	Description	Typical setting	Remarks
Name generation algorithm	The name of the algorithm.		
Input variable N	The name of input variable 1,...,N as configured in the name generation algorithm.	%FirstName% %MiddleName% %LastName%	When the action is executed, the input variables should have a value that is used to calculate the output names.
Output variable M	The name of output variable 1,...,M as configured in the name generation algorithm.	%UserName% %FullName%	
Iteration	A description of the method used to iterate through the name generation algorithm when the action is called multiple times.		

**See also:**

*Script Action: Create User (AD) on page 3*

*Script Action: Create User (no AD) on page 68*

**Variable operations**

*Script Action: Set Variable*

**Function**

Defines and sets a script variable. This action does not do any network calls. It is used for configuration within the User Management script.

## Deployment

Many implementations of User Management scripts are configured in such a way that the input for the script action property values can be specified as a variable. Variables correspond with a column of the input data or they get a fixed constant value in the beginning of the script using this action. For instance, the **Domain** property of the *Script Action: Create User (AD)* on page 3 (and this applies to many other script actions), is usually specified as %Domain% in the script action property value. If all users need to be created in the same domain it is usually easier to specify the name of the domain directly in the script, instead of requiring that the domain name is available in every row of the Project Table.

To specify the contents of a variable directly in the script, this script action should be inserted in the script. It must be inserted prior to any script action that uses this specific variable.

## Properties

Property Name	Description	Remarks
Variable Name	The name of the variable to set	The Name of the variable must be enclosed in "%" characters. e.g. %Domain%
Value	The value of the variable. This value will be used in all following script actions.	The value to which the variable is set. Note that <b>Value</b> might contain the name of another variable or a combination of text and other variables.  The variable can then be used in any following script actions in the script.
Value Type	The type of the variable. That is text, boolean, numeric, date-time or text list.	
Do not show value in windows and log files	Hides the value of the variable. It is no longer shown in log files and windows.	

Resolve variable names in value immediately when action is executed	The method used to resolve variable names in the specified value.	<p>No: Other variable names specified as part of the variable value, are not resolved until the variable is used as an argument in in another script action.</p> <p>Yes: Other variable names specified as part of the variable value are resolved immediately.</p>
---	---	---

*Script Action: Set encrypted variable*

#### Function

This script action sets the value of the specified variable to the encrypted value of the entered text.

#### See also:

*Script Action: Encrypt text on page 563*

*Script Action: Split Variable*

#### Function

Splits the value of an existing variable in two parts, and store the results in two (new) variables. This action does not do any network calls. It is used for configuration and formatting within the User Management script. The Variable is split in two parts, the split position is determined by a separator character available in the data.

#### Deployment

Many implementations of User Management scripts are configured in such a way that the input for the script action property values can be specified using variables. The contents of these variables are usually read from an input data source.

In some cases the value of this variable needs to be manipulated before it can be used to specify a specific script action property. Various functions are available for these kind of operations.



The Split variable script action is used if one specific variable contains information that a certain script action expects to be in different variables. For example, the input data may have a field that contains the variable %HomeDirectory% in the form "server name\share name\sub directory". The *Script Action: Create Directory* on page 341 for instance, requires the name of the server, the name of the share, and the rest of the path to be in three different variables. With the script action **Split variable** it is possible to create the required variables. In this particular example the action is first used to retrieve the server name, and then used again to retrieve the share name.

#### Properties

Property Name	Description	Typical setting	Remarks
Input variable	The name of the variable that contains the information that must be split.		The name of the variable must be enclosed in "%" characters. e.g. %Domain%
Output variable 1	The name of the variable that contains as result the first part of the input string up to the first separator character.		
Output variable 2	The name of the variable that contains as result the rest of the original string		
Result if no split	Specifies which variable contains a copy of the original string as the data cannot be split.	Value of variable 2 empty	Sometimes the data cannot be split if there is no separator character. For such cases, this setting will determine which of the 2 output variables should get a value.

Process from right to left	Specifies that the input string is should be evaluated from right to left	No	if specified, the part of the string after the last separator character is stored in variable 1, and the part before the last separator character is stored in variable 2
Separator(s)	Specifies which character(s) count as separator	No	The variable is split at the first position where one of the specified characters is encountered.

*Script Action: Get variable length*

### Function

Calculates the length (number of characters) of an existing text variable, and stores the resulting number in the specified output variable.

### Deployment

For instance to check if the length of a name is in the range of allowed values.

### Properties

Property Name	Description	Typical setting	Remarks
Input variable	The name of the variable that contains the string of which the length must be calculated		The name of the variable must be enclosed in "%" characters. e.g. %username%
Output variable	The name of the (numeric) variable that will contain the length.	%Length%	

*Script Action: Format Variable Value***Function**

Formats the variable value according to the specified formatting functions. With this action you can specify several formatting functions that are consecutively applied on the value of the input variable. The resulting value is stored in the same variable.

**Deployment**

This script action is typically used to remove undesired characters from the variable value or limit the length of the variable value. The original content of the variable often is determined by a user provided import file. Such a file is likely to contain some irregularities, or the format may be not always be exactly correct. This action helps to correct such problems. For instance, by removing any trailing blanks in the value.

**Properties**

Property Name	Description	Remarks
Variable	The name of the variable that contains the value to be formatted.	The resulting formatted value will be stored in the same variable
Number of Formatting functions Applied	Lists the number of formatting functions that are applied	This is a read only value shown for informational purposes. Double click to open a dialog to configure the formatting functions.
Test Name	A example value that can be specified to test the result of the formatting functions on a value	This value is not used when the script is run

Formatting functions.	A list of formatting functions that are consecutively applied to the value of the variable.	This property is not directly shown in the right pane of the Project. Double click any property to open a dialog that reveals a button to specify the functions used
Do not show results in log files	Enable this option if the input and output values of the formatting functions should not be shown.	Typically used to format password values.

### Specifying Formatting functions

Doubleclicking on any property shows a dialog to configure the properties of this action. Select the **Format functions** button to specify which functions to use and in at order that they are applied.

*Script Action: Update numeric variable*

### Function

This function allows you to perform some basic numerical operations on variables.

### Overview

Operation	Description
Increment value	Increments the variable value by 1. This operation is typically used to accommodate for reiteration within a loop (e.g. as a result of a goto label). Similarly, you could cycle through the rows of a table and use this script action to increment the row counter.
Decrement value	Decreases the variable value by 1. This operation is typically used to accommodate for reiteration within a loop (e.g. as a result of a goto label). Similarly, you could cycle through the rows of a table and use this script action to decrement the row counter.

Add number	Adds a number to an existing numeric variable. The number to add can be held by a variable.
Subtract number	Subtracts a number from an existing numeric variable. The number to subtract can be held by a variable.
Multiply by	Multiplies an existing numeric variable by a number. The number to multiply with can be held by a variable.
Divide by	Divides an existing numeric variable by a number. The number to divide with can be held by a variable.
Convert from variable	Converts an existing text variable to a numeric variable. The variable to convert needs to be selected in the <b>Source variable</b> list box. The name of the new numeric variable needs to be specified in the <b>Numeric variable</b> field.
Convert number to text (format)	<p>Convert an existing numeric variable to a text value, according to the specified format. The format corresponds with the programming C-language printf syntax. For a numeric value of 3168, the following list shows some example values for different format specification:</p> <p>Format specification: %d      resulting text: 3168  Format specification: %06d    resulting text: 003168  Format specification: %x      resulting text: C60  Format specification %08X    resulting text: 00000C60</p>

#### Cycling through the records in a table using Increment / Decrement value

To do this, you would need to define the following:

1. Set a row counter variable (e.g. %RowCounter%) to 0 using the Set variables action item
2. Retrieve the number of rows in a column and assign these to a variable (e.g. %NumberOfRows%) using the option "Get number of rows" in the **Manage table data** script action
3. Create a loop which includes

This loop may be constructed as follows:

1. an If-Then-Else statement (IF the value of %RowCounter% is smaller than %NumberOfRows%, THEN execute a script, ELSE do something else)
2. the action to execute when the IF statement returns TRUE
3. Update numeric variable action to increment the value of %RowCounter% by 1. This will let you cycle to the next row in the table.
4. GoTo statement to return to the If-Then-Else script action to evaluate the next row in the table
5. "No operation" script action as an ELSE clause (IF statement returns FALSE)



#### See also:

*Script Action: If-Then- Else on page 570*

*Script Action: Manage table data on page 528*

*Script Action: Update date-time variable*

#### Function

Performs basic numeric operations on date-time variables.

#### Options

Set value to current date-time



This option allows you to set a date-time variable to the current date. If the variable name already exists, it will be replaced. If it does not exist, it will be created using the specified variable name.

#### Add days-months-years



For this option, you need to specify an existing date-time variable. Days, months and years can be added to an existing date-time variable. This will be needed for instance, if you want to execute an action with a time delay (e.g. removing a disabled user).

#### **Add hours-minutes-seconds**



For this option, you need to specify an existing date-time variable. Hours, minutes and seconds can be added to an existing date-time variable.

#### **Subtract days-months-years**



For this option, you need to specify an existing date-time variable. Days, months and years can be subtracted from an existing date-time variable.

#### **Subtract hours-minutes-seconds**



For this option, you need to specify an existing date-time variable. Hours, minutes and seconds can be subtracted from an existing date-time variable.

#### **Subtract date-time variable**

This function subtracts a date-time value stored in a variable (field: **Date-time variable to subtract**) from another date-time value (field: **Date-time variable**). The result is stored as a numerical value, the difference in seconds in another variable (field: **Result variable (seconds)**).

#### **See also:**

*Script Action: Set Variable* on page 544

*Script Action: Convert value of variable*

### Function

This script action allows you to perform the following variable conversions:

1. Performing a logical AND on the input value and a specified argument
2. Convert a large integer to date-time
3. Converting a large integer to specified text if the value is zero.

### Deployment

This script action is mainly intended to be used in MASS and Automation. In Forms & Delegation, large integers (e.g. lastLogon attribute) are either automatically converted or shown as text (in a form object, these variables will be displayed as text).

### Usage

The name of the input variable should be selected or entered in the Input variable name list box. The name of the converted variable should be selected or entered in the Output variable name list box. If you click the Conversion button, the following dialog box will appear:



Click the **Add** button. The **Data Conversion Routine** dialog box will appear. Under **Conversion Operation** you can select the required conversion type.



### Perform logical AND on the input value and specified argument

The routine Perform logical AND on the input value and specified argument is used to evaluate so called bitmask attribute values. A bitmask attribute is a single attribute that contains multiple properties and property values (e.g. the attribute UserAccountControl).



**Convert a large integer to date-time**

Using this option, you can convert a large integer to date-time (e.g. lastLogon attribute). The large integer will then be converted to a date-time variable (e.g. "127718490668401648" will become "07:51 09/22/2005")

**Converting a large integer to specified text if zero**

In Active Directory, all attributes containing system times, are stored in a large integer (e.g. lastLogon). If this value is equal to zero, a normal conversion to date-time would result in "00:00 01/01/1601". With the conversion routine "Converting a large integer to specified text if zero" you can specify a more meaningful text value to pass.

*Script Action: Convert text to date/time*

**Function**

Converts a text value to a date/time value. Both values are stored in a variable. The method used to convert the text to a date/time value can be specified.

**Deployment**

This action is typically used in UMRA projects to set the account expiration date as specified in a text file when creating new user accounts. In this scenario, a column of the text file contains the user account expiration date and a variable is assigned to this column. This variable stores the expiration date/time as a text string. In order to use this expiration date/time value, it must be converted from a text value to a date/time type value. This can be done with this action. The action takes the text value of the input variable and converts this text to a date/time value type. The resulting value is stored in a variable that can be used to specify the expiration date of a user account.

For the action, the format used to convert the text to a date/time value can be configured.



**Input text variable**

The name of the input variable that stores the date/time text value represented as text.

**Output date/time variable**

The name of the output variable that upon execution of the action stores the date/time value in as a date/time type. The input and output variable names can be the same: In this case the input variable value is overwritten with the result variable value.

**Format**

The format of the input variable text value. The format is specified using the following fields: month,day,year,year,hour,minute and second. In order for this action to succeed, the format must correspond with the input variable value. Example: When the date/time is specified as 7/27/2005 14:30 the format must be specified as: month/day/year hour:minute. A number of predefined formats can be selected from the list. Note that not all fields need to be specified. In this case, the action will use the default values.

**Default values - Format field - Default value**

Select one of the possible format fields from the list Format field and specify the default value in the edit box Default value. The default value is only used when it is not specified in the Format string.

**See also:**

*Script Action: Create User (AD) on page 3*

*Script Action: Create User (no AD) on page 68*

*Script Action: Convert to multi-value variable*

**Function**

Converts the value of a variable into multiple values. The multiple values will be stored under one variable. This action does not do any network calls. It is used for configuration and formatting within the User Management script. The Variable is split in multiple values and stored

under a new variable, the split position is determined by a separator character available in the data.

### Deployment

Many implementations of User Management scripts are configured so that the input for the properties of the script actions they contain are defined as variables. The contents of these variables are usually read from the input data. This table is often created from information provided by the user. This information may however not fit seamless to the requirements of the script actions in the script. Therefore there are several functions that can be used in the script for some formatting of data. This is one of them.

This function in particular can be used when one variable in the input data contains information that has multiple values for one property of a script action. For example, the input data may have a field that contains the variable %GroupMemberships% specified as "Domain\Administrators;Domain\Backup Operators;Domain\Users". The action *Script Action: Set group membership (AD)* on page 135 for instance can be used to make an user account member of multiple groups. With the script action *Convert to multi-value variable* it is possible to create the required variable which contains the three groups as separate values under one variable. In this particular example the action is used to separate the value of a variable into three new values for an other variable.

### Properties

Property Name	Description	Typical setting	Remarks
Input variable:	The variable that contains the data which is multi-value.		The data in the input variable should be separated by an separator character.
Output variable:	The variable which contains the same data as the input variable only now the data is available as multiple values instead of one value.		The output variable can be a different variable or the same variable as the input variable.

Separator character	A character that separates one value from the other. This character is used to determine where the new value begins.	;	The following characters can be chosen to separate the values: , ; <tab>
Insert empty values	When two separator characters are placed directly after each other a blank value could be created.	No	When you want blank values to be defined this property should be set to 'Yes'. When you want to remove all blank characters this property should be set 'No'.

*Script Action: Manage multi-text value variable*

### Function

Manages a multi-text value variable. Enables you to sort a multi-text value variable and delete empty text values.

### Deployment

This script action is used to manage the multi-text value that is created with *Script Action: Convert to multi-value variable* on page 556.

### Properties

Property Name	Description	Typical setting	Remarks
Variable	The variable that contains the data which is converted to multi- value.		Use <i>Script Action: Convert to multi-value variable</i> on page 556 to convert a data string to a multi value variable.
Delete empty text values	When a multi-text value contains empty values, these empty values can be deleted by setting this property to 'Yes'.	No	

Sort values in ascending order	Sorts the multi-text value in ascending order	No	
Sort values in descending order	Sorts the multi-text value in descending order	No	

*Script Action: Merge multi-text variable values*

### Function

This action merges two input variables into one output variable

Name	Description
Input variable 1	Name of input variable 1
Input variable 2	Name of input variable 2
Output variable	Name of the output variable
Delete duplicates	Specifies that duplicate entries in the content of the output variable must be deleted
Delete empty values	Specifies that empty values in the content of the output variable must be deleted
Sort	Specifies that the content of the output variable must be sorted

### Deployment

This action is typically used in a script where you need to merge two tables and clean up the content.

*Script Action: Export Variables*

### Function

Writes the value of one or more variables to a text file.

## Deployment

Typically used at the end of a script to record the results of the script operation. For instance, in a script that creates user accounts often the user logon name and the password are exported to a file, so the user can be informed. Especially essential when the account are created with random passwords.

## Properties

Doubleclick on any property to open a special dialog to set all properties from one window.

Property Name	Description	Typical setting	Remarks
Number of exported variables	The number of exported variables.		Read only property.
Export file name	The name of the file		The name of the file can contain variables: %NowDay% : The current day (00,...,31) %NowMonth%: The current month (01,...,12) %NowYear%: The current year (2005,...)
Exported text fields	A list of strings that are written to the Export file. In order to export variables, specify the variable name in the export string.		Examples of a export strings: Created user %Username% in domain %Domain% %Username% %Password%  All strings will be exported on the same line in the file. If the output must be on more lines, use a separate <b>Export Variables</b> script action.
Field separator	The character that is exported to separate the exported fields.	,	

Value separator	The character used to separate multi-values	, or ;	The character is inserted between the values of a multi-value variable value. In order to be able to distinguish the different values of a multi-value variable, the values should not contain the value separator character. Example: When exporting the <b>memberOf</b> attribute of user accounts, the object distinguished names of the groups of which the user account is a member are returned. These names contain comma's: CN=GroupA, DC=domain (note the comma between GroupA and DC). To separate multiple groups in this case, another value separator should be used, for instance a semi-colon (;).
Enclose fields with blanks	Specifies that fields that contain blanks will be enclosed by the enclose character.		
Enclose character	Specifies the character that is inserted around strings that contain blank chars.	"	
UNICODE format	When checked, the exported data is save in UNICODE format.		

*Script Action: Delete variable***Function**

Deletes a specific variable from the list with variables. If a the value of a variable is no longer valid, it might be a good idea to delete the variable so it can not accidentally be used in subsequent script actions.

**Deployment**

The execution of certain actions might invalidate the value of certain variables. To prevent incorrect usage of these variables, the variables can be deleted with this action. Example: Suppose a script is used to move a user account from one domain to another domain in the same forest. The target user account has a number of properties, for instance the Security Identifier (SID). The SID can be used in User Management to set up directory permissions. Now suppose the user account must be moved, and then the home directory must be moved to another location. For the new home directory, permissions must be setup. The script actions involved are:

<b>Script action</b>	<b>Description</b>
<i>Script Action: Get user (AD)</i> on page 31	Binds to the user account. The user account and the security identifier are exported in variables (%UserObject%, (%UserSid%))
<i>Script Action: Move - rename user (AD)</i> on page 63	Moves the user account to the new domain in the same forest. The variable %UserSid% now becomes invalid since a new Security Identifier is generated for the moved user account.
<i>Script Action: Copy directory</i> on page 347	Copies the original home directory to the new location and setup the new security settings using variable %UserSid%.
<i>Script Action: Delete directory</i> on page 357	Deletes the original home directory.

In this example, the variable %UserSid% is no longer valid, once the user is moved and cannot be used to setup the new security settings for the target home directory. Instead, the variable %UserSid% should be deleted. Then a new **Get user (AD)** action should be used to determine the new value of the SID.



In more complicated scripts, you might want to delay certain operations. For instance before an operation is retried or to limit network load. For this purpose, this action can be used. The action simply suspends the script for the specified time. Note that during this time, the script cannot be aborted.

**Properties**

Property Name	Description	Typical setting	Remarks
Variable	The name of the variable to be deleted.		

*Script action: Delete multiple variables*

**Function**

Delete one or more variables from the project variable list.

**Properties**

Property Name	Description	Typical setting	Remarks
Variable names	The names of one or more variables that must be deleted from the variable list (example: %FirstName%, %MiddleName%, %LastName%).		The names must be separated with comma's (,).

*Script Action: Encrypt text*

**Function**

Encrypts the text data of one variable and stores the result in another variable. The same name can be used for both input and output text.

**See also:**

*Script Action: Set encrypted variable on page 546*

*Script Action: Generate random number*

**Function**

Generates a random number and assign the value to a variable. The name of the variable and the minimum and maximum possible values of the random number can be specified.

**Deployment**

The action is used to generated random values. The random value is generated as a number. The minimum and maximum possible values can be specified. These limits are included, e.g. the generated number can be equal to the value of these limits. The number is assigned to a variable as a numeric value. This variable can be used in other variables, also text variables, for instance to create a random user id number as described below:

1. Generate variable **%ID%** as a random number in the range 0,...,99.
2. Specify variable **%UserID%** as text value: **User%ID%**. See *Script Action: Set Variable* on page 544 for more information on how to do this.
3. Now if number 47 is generated (**%ID%=47**) the resulting variable equals User47 (**%UserID%=User47**)

**Properties**

Property Name	Description	Typical setting
Variable name	The name of the variable that stores the generated random number.	
Minimum value	The minimum possible value of the generated number.	0

Maximum value	The maximum possible value of the generated number.	999
---------------	---	-----

**See also:**

*Script Action: Set Variable* on page 544

*Script Action: Generate password*

**Function**

This script action generates a password according to the specified algorithm and stores the result in the variable %Password%.

*Script Action: Log Variables*

**Function**

Writes the current value of all variables to the User Management log file. Note that is not the same as the *Script Action: Export Variables* on page 559 which is used to export the value of variables to a text file. It can be used at several positions in the same script to log the values of the variables at the moment the specific line of the script was executed.

**Deployment**

Typically used for script debugging purposes when developing or customizing a User Management script. If you want to output specific variables to a file for reviewing or post processing outside User Management use the *Script Action: Export Variables* on page 559.

**Properties**

This action has no configurable properties

**See also:**

*Script action: Log specific variables* on page 566

*Script Action: Log Specific Variables***Function**

Writes the current value of the specified variables to the User Management log file. Note that is not the same as the *Script Action: Export Variables* on page 559 which is used to export the value of variables to a text file. It can be used at several positions in the same script to log the values of the variables at the moment the specific line of the script was executed.

**Deployment**

Typically used for script debugging purposes when developing or customizing a User Management script. If you want to output specific variables to a file for reviewing or post processing outside User Management use the *Script Action: Export Variables* on page 559.

**Properties**

Property Name	Description	Typical setting	Remarks
Number of logged variables	List the currently configured number of variables to log		Readonly. Double click to open a dialog to specify the names of the variables that should be logged.

**See Also:**

:*Script Action: Log Variables* on page 565

*Script Action: Get session variable*

The action retrieves a variable from the *session* on page 81 variable list and copies or updates the value to the project variable list. The variable name must be specified. Example: %ConnectionString%. When the variable does not exist in the session variable list, an error is generated. If the variable already exists in the project variable list, the existing value is overwritten.

*Script Action: Set session variable*

The action stores a variable value in the *session* on page 81 variable list. The variable name must be specified. Example: %ConnectionString%. When the variable does not exist in the project variable list, an error is generated. If the variable already exists in the session variable list, the existing value is overwritten.

*Script Action: Check session variable*

The action checks if a variable exist in the *session* on page 81 variable list. The variable name must be specified. Example: %ConnectionString%. The action returns the result in output property **Variable exists flag**.

*Script Action: Delete session variable*

The action deletes a variable from the *session* on page 81 variable list. The variable name must be specified. Example: %ConnectionString%. When the variable does not exist in the session variable list, an error is generated.

#### 4.3.12. Programming

**Script Action: Map variable****Function**

This function **maps** the value of an input variable to a value of an output variable. The mapping table specifies the value of the output variable for each possible value of the input variable.

**Deployment**

This script action is usually used in a scripts to handle the case of exceptions to the main rule of the script. The output variable can also be used as a label, .e.g. as the target of a **GOTO** action.

Example: A particular script that creates a user account uses the variable %HomeServer% to contain the home server of the new account. This name is later in the script used to specify the home directory of the user: %HomeDirectory%=%HomeServer%\users\%UserName% by means of the *Script Action: Set Variable* on page 544. Now this setting works fine for most home servers in your network, but for a particular server, the location where the home directory should be created is different: For your home server named **OAK** you want the home directory of the user to be %HomeServer%\students\%UserName%

In the above case, you can use the **map variable** action. You specify the variable %HomeServer% to be the **input variable**, and the variable %HomeDirectory% as the **output variable**. In the mapping table you specify **OAK** as the input value to match and %HomeServer%\students\%UserName% as the associated value. The result is that whenever the home server is **OAK** the name of the home directory is changed from %HomeServer%\users\%UserName% to %HomeServer%\students\%UserName%.

#### Properties

Property Name	Description	Typical setting	Remarks
Input variable	The name of the variable that contains the information that must be looked up in the list.		The Name of the variable must be enclosed in "%" characters. e.g. %Domain%.
Output variable	The name of the variable that is modified by this script action.		The Name of the variable must be enclosed in "%" characters. e.g. %Domain%.
Number of Input-Output values	The number of entries in the Mapping table.		This is only shown in the property list, not in the configuration dialog.

Mapping Table	Specifies which input value results in the specified output value.		Specifies a list of (input value, output value) pairs. If the contents of the input variable matches the input value in the list, the output variable will be set to the corresponding output value.  This is only shown in the configuration dialog, not in the properties list itself.
Set output variable to default value if no match found.	If set to Yes, then, when no match is found in the mapping table, the output variable is to the below specified default value. If set to NO, and no match if found, the output variable is not altered.		
Default value of output variable	Specifies the value the output variable gets when there is no match.		This value is only used when the "Set output variable to default value if no match found" flag is set to Yes.
Case sensitive compare	Specifies if the compare function to find a match must be case sensitive.	No	

**Script Action: Go to Label****Function**

Unconditionally jumps to the script action with a specified label, skipping all actions between this action and the action with the specified label.

### Deployment

Each script action can jump to an other labeled action in case of an error. In combination with *Script Action: Map variable* on page 567, you can conditionally specify the value of a label to jump to.

### Properties

Property Name	Description	Remarks
Label	The label of the script action that is to be executed directly after this action.	Note that the name of the destination label may contain variables. This makes it possible to perform conditional jumps if required, for instance by using <i>Script Action: Map variable</i> on page 567 earlier in the script to create a variable that specifies a label.

To setup the label of an script action, select the target action in the script section (lower left area) of the project window. Right click the mouse or select main menu option Actions and select Set script action label.

### Script Action: If-Then- Else

#### Function

Evaluates a condition and then performs one or more script actions depending on the results of that condition.

#### General

An IF-THEN-ELSE statement is essential in the world of scripting and programming. It evaluates a condition and then performs an action depending on the outcome of the evaluation. Every IF-THEN- ELSE statement follows the same structure:



IF the result is TRUE, then execute action A, else (the result is FALSE) execute action B. The screenshot below illustrates this principle. In the IF section, the evaluation criterion is specified. If the variable %CurrentDate% is equal to 1 January 2006, THEN the action labeled as NEW YEAR is executed. ELSE (the variable %CurrentDate% is not equal to 1 January 2006), the action labeled as NOT NEW YEAR is executed.



For the IF section there are many different evaluation conditions available.

### **Script Action: Execute script**

#### **Function**

This script action executes the script of another project. When the script action has been executed, the variables in project B will be updated and merged into project A.

#### **Deployment**

There are many situations where this script can be employed:

- to reuse temporary variables. These are variables which only contain references to data and not the actual data itself. One example would be a wizard containing multiple project forms where an LDAP session is established for each form. As soon as the 1st LDAP session is closed, the variable containing the session no longer exists. Using the **Execute script** script action, the LDAP session script generating the LDAP session variable can be called from within any project form in the Wizard.
- to reuse complex scripts. Some scripts can take a lot of time to develop and this script action offers the possibility to reuse it for other projects;
- to initialize scripts (e.g. to change the user environment settings so that a project can easily be used in other user environments).

The project of which you want to execute the script can be selected in the **Project name** list box.

### Script Action: For-Each

#### Function

Steps through the specified table. For each row in the specified table, a script is executed, using the values of the current row.

#### Deployment

Typically, this is used if the same set of actions must be done on multiple objects.

For instance, earlier actions in current script may have read a table of user names from a file, and an other UMRA script may have been configured to create a single new user, given the name of the user in a variable. This action can then be used to call that script, so that a user is created for each name in the file.

The following properties are shown. Double click on any of them to open a special configuration dialog.

Table variable name	Variable name of the table on which the For-Each action has to operate
Script project	Name of the project which contains the script which needs to be executed for each row of the table
Column x	Specifies the name of the variable of the called script that must be filled with the value in the specified column of the table
Script project input variables	Specifies which variables in the current script are passed on to the called script. See <i>For each - Input variables</i> on page 573 for more information

Return variable	Specifies the name of the variable that must be returned when the called script has finished.
Stop loop on error	Here you can specify if the script should continue upon encountering an error or not
Break conditior	Specifies a condition that is checked after each call of the script.If true,the rest of the table will be skipped, and execution resumes with the next action in the current script

**See also:**

*Script Action: Manage table data on page 528*

*For each - Input variables*

**Pass variable method specification**

These options specify which variables that are available in the current script are passed on to the project that is called repeatedly.

Passing on values of variables to the called project can cause considerable overhead in specific circumstances, especially when some variables contain a lot of data, such as an table with many entries. Therefore these options let you control which variables to pass, and which not, in order to enhance performance.

**Pass all variables**

All variables that are available in the current script are passed on to the repeatedly called project. This means that all variables that are defined in the current script are available for the called project. Use this if you do not know which variables you will actually use in the called script, or if they do not contain large amounts of data.

This is the default setting.

**Pass "for each" variables only**

Only the variables that are bound to a specific column in the table over which the "for each" is performed, as specified on the "fore each" tab, are passed on.

This gives the best performance. Use this if you do not want to make available any other variables from the current script in the called project.

**Pass "for each" variables and the specified variables**

In addition to the "for each" variables, the variables specified in the specified Variable list are also passed on to the called script. Use this option if you want to use some specific additional variables from the current script in the called script.

**Pass all but the specified variables**

All variables are passed to the called script, except those specified in the Variable list. Use this if you want access to most variables in the called project, but there are some large variables you do not need in the called project.

A variable that often does not need to be passed to the called project is the "table" variable, that was used to generate the "for each" variables. As the called project usually does not need access to the entire table, but only to the generated variables that represent the current row in the table, the original "table" variable can be added to the list of excluded variables.

**Variable list**

This list contains the variables that are explicitly passed on when the option '**Pass "for each" variables and the specified variables**' is chosen

The list contains the variables that are explicitly not passed on when the option '**Pass all but the specified variables**' is chosen.

It is not used with the other options.

**Script Action: Delay****Function**

Waits for a specified number of milli-seconds (1000 milli-seconds = 1 second). This action can be used for instance before the retry of a failed action.

**Deployment**

In more complicated scripts, you might want to delay certain operations. For instance before an operation is retried or to limit network load. For this purpose, this action can be used. The action simply suspends the script for the specified time. Note that during this time, the script cannot be aborted.

**Properties**

Property Name	Description
Delay (milli- seconds)	The delay specified in milli-seconds.

**Script Action: No operation****Function**

No operation. The script action does not execute any operation. The script will continue immediately with the next action.

**Deployment**

This action is typically used as a reference point, e.g. as the target of a *Script Action: Go to Label* on page 569 script action.

**See also:**

*Script Action: Go to Label* on page 569

**4.3.13. Mail****Script Action: Send mail message****Function**

This action specifies the parameters for accessing the mail server and sending an e-mail message.

Property Name	Description
To:	E-mail recipient
Cc:	Cc is an abbreviation for carbon copy. If you add a recipient's name to this box in a message, a copy of the message is sent to that recipient
From:	Name of the sender
Subject:	Subject line for the e-mail message
Message:	Body text for the e-mail message
X-Sender:	Some mail software expect 'Sender:' to be an e-mail address which you can send mail to. However, some mail software has as the best authenticated sender a POP or IMAP account, which you might not be able to send to. Because of this, some mail software put the POP or IMAP account into an X-sender header field instead of a Sender header field, to indicate that you may not be able to send e-mail to this address.
Mail Server:	The mail server which handles mail to addresses in the domain associated with the mail server
Mail Server Port:	
Authentication:	
Username:	
Password:	

### Deployment

This action is typically used in a script where an e-mail needs to be sent as a result of a previous action.

### Script Action: Send HTML mail message

#### Function

This action is used to send an HTML E-mail message. Optionally, the mail message can contain one or more attachment files.

Property	Description
SMTP server	The DNS name or IP address of the SMTP server to be used by UMRA to send the e-mail (example: mail.company.com). Optionally, the SMTP port can be specified (mail.company.com:25). By default, port number 25 is used.
To	The recipient of the e-mail message. To specify multiple recipients, use a semi-colon (;) to separate the recipients (example: john@domain.com; william@company.com).
Cc	The recipients that should receive a copy of the e-mail message. To specify multiple recipients, use a semi-colon (;) to separate the copied recipients (example: john@domain.com; william@company.com).
Bcc	The recipients that should receive a blind copy of the e-mail message. To specify multiple recipients, use a semi-colon (;) to separate the copied recipients (example: john@domain.com; william@company.com).
From	The name of the sender of the e-mail message (example: john@company.com).
Subject:	The subject of the e-mail message (example: User creation report).
HTML file	The name of the file that contains the HTML contents of the e-mail message. Specify either this property or property <b>HTML text</b> . The HTML contents of the file can contain UMRA variables. If the HTML contents contains links to files, the files should be contained in the same directory as the file or in the directory specified by property <b>HTML link directory</b> . (example: D:\UMRA\EmailTemplates\UserCreationReport.html)
HTML text	The HTML contents of the e-mail message. Specify either this property or property <b>HTML file</b> . The HTML contents can contain UMRA variables. If the HTML contents contains links to files, these files should be contained in the directory specified with property <b>HTML link directory</b> (example: <HTML> ... </HTML>).
HTML link directory	The directory that contains the files referred to in the HTML contents, specified with property <b>HTML text</b> or <b>HTML file</b> (example: D:\UMRA\EmailTemplates\images).

Attachments	The name of one or more files to be attached to the e-mail message. To specify multiple attachments, use the semi-colon (;) as a separator character (example: D:\UMRA\Data\UserLog.txt).
-------------	---

**Remarks**

1. All properties can contain variable names. At runtime, these variable names are replaced with their actual values.
2. The content of the E-mail message is specified by property **HTML file** or by property **HTML text**. Only one of these properties can be specified.
3. If the HTML contents contains links to for instance image files, these files should be stored in one and the same directory. The directory is specified with property **HTML link directory**. If property **HTML file** is used to specify the HTML contents and property **HTML link directory** is not specified, the files should be contained in the same directory as the HTML file. The references to linked file should not contain a directory name.

**4.3.14. Powershell****Active Directory permissions**

*Script Action: Get AD permissions*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

**Function**

Retrieve permissions of an Active Directory object, like a group, computer, user or organizational unit. The action returns a table with rows for each permission on that object.

**Properties**



Property Name	Description	Typical setting
Identity	The Identity parameter identifies the Active Directory object (group, user, computer or organizational unit) you want to get the permissions of. Use for example the distinguished name (OU=sales,DC=tools4ever,DC=com), the domain\account name (tools4ever.com\Jsmith) or the GUID.	CN=jsmith,OU=sales,DC=tools4ever,DC=com tools4ever.com\Jsmith JSmith JSmith@tools4ever.com or the GUID.
Filter account	The 'Filter account' parameter identifies the account (user or group etc.) of the permissions you want to receive. Use for example the distinguished name (CN=jsmith,OU=sales,DC=tools4ever,DC=com), the domain\account name (tools4ever.com\Jsmith), the User principal name (UPN). (JSmith@tools4ever.com) or the GUID. The 'Filter account' parameter is used to filter the results.	
Include inherited permissions	Optional value. Set the value of this parameter to 'Yes' to show the inherited permissions as well.	

Where clause	Optional value. Use this property to filter the results. Use the following syntax: Where-Object {\$_NameOfProperty -eq 'value'}. For example: Where-Object {\$_User -eq 'John Smith'}. To use different operators, like -ne, -like etc. read the powershell documentation. Use the   operator to separate more Where clauses.	
Domain controller	Optional value. The name of the domain controller used to retrieve the Active Directory object's permissions from. Use the fully qualified domain name (FQDN) of the domain controller you want to use. For example: EXCHSERVER.tools4ever.com.	EXCHSERVER.tools4ever.com
ADPermissionsTable	The resulting table with columns for Identity, AccessRights, User, Deny, ChildObjectTypes, ExtendedRights, InheritanceType, InheritedObjectType and Properties for each ACE.	

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name
ADPermissionsTable	The resulting table with the columns described below.	%ADPermissionsTable%

**Table: Permissions table contents**

Column name	Description	Example
Identity	The name of the object that you are getting permissions from.	domain.com/Users/John Smith
AccessRights	A description of the access right. Possible values: CreateChild, DeleteChild, ListChildren, Self, ReadProperty, WriteProperty, DeleteTree, ListObject, ExtendedRight, Delete, ReadControl, GenericExecute, GenericWrite, GenericRead, WriteDacl, WriteOwner, GenericAll, Synchronize, AccessSystemSecurity.	GenericRead
User	The account for which the permission applies	BUILTIN\Administrators

Deny	A flag indicating if the permission is granted or denied.	False
ChildObjectTypes	The ChildObjectTypes parameter specifies what type of object the permission is with.	
ExtendedRights	The ExtendedRights parameter specifies the extended rights needed to perform the operation. Valid values include: Send-As, Receive-As, View Information Store status	
InheritanceType	The InheritanceType parameter specifies whether permissions are inherited	
InheritanceObjectType	The InheritedObjectType parameter specifies what kind of object inherits this ACE	
Properties	The Properties parameter specifies what properties the object contains.	

*Script Action: Add AD permission*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

**Function**

Adds a permission to an Active Directory object, like a group, computer, user or organizational unit. Use the parameters to specify the correct permission.

**Properties**

Property Name	Description	Typical setting	Remarks
Identity	Use this parameter to specify the identity of the object that you are getting permissions on.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	

Account	The Account parameter identifies the user, group or computer of the permission you want to add to the Active Directory object. Use for example the distinguished name, the domain\account name, the User principal name (UPN) or the GUID.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\Jsmith, JSmith@tools4ever.com	
Access rights	Optional value. The 'Access rights' parameter identifies the access rights the account gets on the Active Directory object. If this property is not specified, the 'Extended rights' property must be specified	Delete, ReadControl, WriteDacl, WriteOwner, Synchronize, AccessSystemSecurity, GenericRead, GenericWrite, GenericExecute, GenericAll, CreateChild, DeleteChild, ListChildren, Self, ReadProperty, WriteProperty, DeleteTree, ListObject, ExtendedRight	

Extended rights	Optional value. The 'Extended rights' parameter identifies the extended rights the account gets on the Active Directory object. If this property is not specified, the 'Access rights' property must be specified	Open-Address-Book, Allowed-To-Authenticate, Receive-As, Send-As, User-Change-Password, User-Force-Change-Password, Send-To, Apply-Group-Policy, Allowed-To-Authenticate, Enable-Per-User-Reversibly-Encrypted-Password.	
Child object types	Optional value. The 'Child object types' parameter specifies what type of object the permissions is with. This parameter can only be specified when you specify the CreateChild or DeleteChild permission in the 'Access rights' property.		

Inheritance type	Optional value. The 'Inheritance type' parameter specifies whether permissions are inherited.	None, All, Descendants, SelfAndChildren, Children	
Inherited object type	Optional value. The 'Inherited object type' parameter specifies what kind of object inherits this ACE. Use for example: User, Group, Computer etc..	User, Group, Computer etc.	



Properties	Optional value. The 'Properties' parameter specifies on what properties of the Active Directory object this ACE applies to. This parameter can only be used when the ReadProperty, WriteProperty or Self permission is used in the 'Access rights' property.	E-mail-Addresses, Display-Name, Exchange-Information etc.	
Deny	Optional value. Set the 'Deny' parameter to 'Yes' to deny the permission for the specified account on this Active Directory object.	Yes, No	

Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller you want to use.	EXCHSERVER.tools4ever.com	
-------------------	---	---------------------------	--

*Script Action: Remove AD permission*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

### Function

Remove a permission of an Active Directory object, like a group, computer, user or organizational unit. Use the parameters to specify the permission you want to remove.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the Active Directory object (group, user, computer or organizational unit) you want to remove a permission from.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith, JSmith@tools4ever.com or the GUID.	
Account	The Account parameter identifies the user account or group of which you want to remove permissions of this Active Directory object from. Use for example the distinguished name, the domain\account name, the User principal name (UPN) or the GUID.	CN=jsmith,OU=sales,DC=tools4ever,DC=com, tools4ever.com\Jsmith, JSmith@tools4ever.com	

Access rights	Optional value. The 'Access rights' parameter identifies the access rights you want to remove from the Active Directory object. If not specified, and the 'Extended rights' property is not specified, all permissions for that account will be removed.	Delete, ReadControl, WriteDacl, WriteOwner, Synchronize, AccessSystemSecurity, GenericRead, GenericWrite, GenericExecute, GenericAll, CreateChild, DeleteChild, ListChildren, Self, ReadProperty, WriteProperty, DeleteTree, ListObject, ExtendedRight	
Extended rights	Optional value. The 'Extended rights' parameter identifies the extended rights you want to remove from the Active Directory object. If not specified, and the 'Access rights' property is not specified, all permissions for that account will be removed.	Open-Address-Book, Allowed-To-Authenticate, Receive-As, Send-As, User-Change-Password, User-Force-Change-Password, Send-To, Apply-Group-Policy, Allowed-To-Authenticate, Enable-Per-User-Reversibly-Encrypted-Password.	

Child object types	Optional value. The 'Child object types' parameter specifies what type of object the permissions is with. This parameter can only be specified when you specify the CreateChild or DeleteChild permission in the 'Access rights' property.		
Inheritance type	Optional value. The 'Inheritance type' parameter specifies whether permissions are inherited.	None, All, Descendents, SelfAndChildren, Children	

Inherited object type	Optional value. The 'Inherited object type' parameter specifies what kind of object inherits this ACE. Use for example: User, Group, Computer etc..	User, Group, Computer etc.	
Properties	Optional value. The 'Properties' parameter specifies on what properties of the Active Directory object this ACE applies to. This parameter can only be used when the ReadProperty, WriteProperty or Self permission is used in the 'Access rights' property.	E-mail-Addresses, Display-Name, Exchange-Information etc.	

Deny	Optional value. Set the 'Deny' parameter to 'Yes' to deny the permission for the specified account on this Active Directory object.	Yes, No	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller you want to use.	EXCHSERVER.tools4ever.com	

*Script Action: Set AD permissions (advanced)*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

#### Function

Set permissions of an Active Directory object, for example, a group, computer, user or organizational unit. The action has a table with rows for each permission as input and overwrites all existing permissions for that object.

## Deployment

The 'AD permissions table' parameter is an UMRA table with one column. This table has rows for each permission and each row in that column contains the last part of the 'Add-ADPermission' commandlet. The first part of the commandlet is generated with the Identity parameter and looks like 'Add-ADPermission -Identity "John Doe"'. The second part contains for example: -AccessRights Read -User "Jane Doe" - . To get this table, first retrieve a table with the UMRA action 'Get AD permissions'. Optionally, you can edit the table. Use this table as input for the UMRA Form 'ConvertToADPermissionTable'. This form, and the get and set example forms, are located in the 'Example projects' folder in the UMRA installation directory.

## Properties

Property Name	Description	Typical setting	Remarks
Identity	Use this parameter to specify the identity of the object that you are getting permissions on.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith@tools4ever.com or the GUID.	
AD permissions table	This parameter is used to set all the permissions of a Active Directory object. Retrieve a permissions table by using the ' Get AD permission' and filter the results with the UMRA project: ConvertToADPermissionTable.	%ADPermissionsTable%	



Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller you want to use.	EXCHSERVER.tools4ever.com	
-------------------	---	---------------------------	--

*Script Action: Get owner*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

### Function

Get the owner of an Active Directory object, for example, a group, computer, user or organizational unit. The action returns a table with a row for the owner of that object.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	Use this parameter to specify the identity of the object that you are getting permissions on.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\jsmith, JSmith@tools4ever.com or the GUID.	

Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller you want to use.	EXCHSERVER.tools4ever.com	
-------------------	---	---------------------------	--

### Output Properties

When the action is run, the actual value of the properties are determined at run time, and the action is executed using these values. Generally these values are not stored for later usage.

By default the following properties are saved in a variable for usage in other scripts. Properties that are exported are shown with an image with a green arrow in the properties list.

Property	Description	Default variable name	Remarks
ADOwnerTable	The resulting table with one row and columns for Identity, Owner, IsValid and ObjectState, describing the owner of the Active Directory object.	%ADOwnerTable%	

*Script Action: Set owner*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

#### Function

Set the owner of an Active Directory object, for example, a group, computer, user or organizational unit. The action has a table with rows for each permission as input and overwrites all existing permissions for that object.

#### Properties

Property Name	Description	Typical setting	Remarks
Identity	Use this parameter to specify the identity of the object that you are getting permissions on.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\jsmith, JSmith@tools4ever.com or the GUID.	
Owner	The Owner parameter identifies the new owner of the Active Directory. Use for example the distinguished name, the domain\account name or the GUID.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\jsmith, JSmith@tools4ever.com or the GUID.	
Domain controller	Optional value. The name of the domain controller that retrieves data from Active Directory. Use the fully qualified domain name (FQDN) of the domain controller you want to use.	EXCHSERVER.tools4ever.com	

## Group management

*Script Action: Set Managed By*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

### Function

Set the **Managed By** attribute of the specified group. The action updates the **Managed By** attribute in Active Directory and optionally updates the Active Directory permissions to allow the specified manager to update the membership list (**Manager can update membership list**).

### Deployment

This action updates the **Managed By** setting of an existing group.

### Properties

Property Name	Description	Typical setting	Remarks
Identity	Identifies the group you want to set the 'Managed By' option of. Use for example the distinguished name, the domain\account name, or the GUID.	CN=group_a,OU=sales,DC=tools4ever,DC=com  tools4ever.com\group_a	

User name	The parameter specifies the name of the user, group, or contact that appears in the Managed by tab of the Active Directory object. You can use any of the following values for this parameter: Distinguished name (DN), canonical name, GUID, Name or Display name.	(CN=jsmith,OU=sales,DC=tools4ever,DC=com), tools4ever.com\Jsmith, JSmith@tools4ever.com or the GUID.	
Allow update membership list	Set this parameter to 'Yes' to allow the manager (the user that has the Managed By' option of this group) to update the membership list.	Yes or No	
Domain controller	Optional value. The name of the domain controller used to access and update the group. Use the fully qualified domain name (FQDN) of the domain controller you want to use. For example: domaincontroller.tools4ever.com. This property is used to prevent replication problems.	domaincontroller.tools4ever	

*Script Action: Get (nested) group memberships*

Note: This action requires the UMRA Powershell Agent service to be installed with the Exchange 2007 Management Tools. See ***Manage Active Directory with the UMRA Powershell Agent service*** for more information.

**Function**

Get the group memberships of an Active Directory object. Like for example: a group, user or computer. The action generates a table that hold the names of the groups, including nested groups, of which the Active Directory object is a member. This action retrieves the group memberships of an existing Active Directory object.

**Deployment**

This action retrieves the group memberships of an existing Active Directory object.

**Properties**

Property Name	Description	Typical setting	Remarks
Identity	The Identity parameter identifies the Active Directory object (group, user, computer or organizational unit) you want to get the group memberships of. Use the distinguished name to specify this parameter (CN=jsmith,OU=sales,DC=tools4ever,DC=com).	CN=jsmith,OU=sales,DC=tools4ever,DC=com	

Group table	The table of groups that the object is a member of. Each group is specified with the format Domain\GroupAccount.	%GroupTable%	The property is an output only property, specified as a variable.
-------------	--	--------------	---

### File system

*Script Action: Get disk space*

### Function

The action gets disk space information from a specified computer. The information is returned in a table and output variables holding minimum and maximum values.

### Properties

Property Name	Description	Typical setting	Remarks
Computer name	The name of the computer of which to receive its disk spaces. If this parameter is not specified, the local machine is used. Use one of the following formats: NETBIOS (zeus), FQDN (zeus.tools4ever.com) or TCP/IP address (192.168.196.87).		Examples: zeus zeus.tools4ever.com 192.168.196.87



Disk space table	When specified, the output variable stores the disk information in a table.	%DiskSpaceTable%	The table contains the following columns: Name (C:) VolumeName (UserData) FreeSpace (in bytes) Size (in bytes) Description (Local fixed disk) __SERVER (ZEUS) DriveType. See the Remarks section for more information on the DriveType parameter.
Minimum free disk space MB	When specified, the output variable stores the available disk space in MB of the logical disk drive with the minimum disk space available. For this property, only hard disks are taken into account.	%MinFreeDiskSpaceMB%	
Minimum free disk space drive	When specified, the output variable stores the logical disk drive (example: D:) with the minimum disk space available. For this property, only hard disks are taken into account.	%MinFreeDiskSpaceDrive%	

Maximum free disk space MB	When specified, the output variable stores the available disk space in MB of the logical disk drive with the maximum disk space available. For this property, only hard disks are taken into account.	%MaxFreeDiskSpaceMB%	
Maximum free disk space drive	When specified, the output variable stores the logical disk drive (example: D:) with the maximum disk space available. For this property, only hard disks are taken into account.	%MaxFreeDiskSpaceDrive%	

#### Remarks

The Disk space table returns in one column the drive type. This numerical value corresponds to the type of disk drive the logical disk represents. The values have the following meaning:

DriveType value	Meaning
0	Unknown
1	No Root Directory
2	Removable Disk
3	Local Disk
4	Network Drive
5	Compact Disc
6	RAM Disk

**Active Directory utility***Script Action: Get PDC (AD)***Function**

The action gets the Fully Qualified Domain Name of the computer holding the PDC Operations Master FSMO role on the current domain. The name is returned in an output variable, specified by default as %PDCDomainController%.

**Agent service session***Script Action: Setup Powershell Agent service session*

The action is used to initialize a Powershell Agent service session. A number of UMRA Powershell actions use a Powershell session. This action is used to create such a session. The action has only a single output property: **Session ID**. The property should be stored in an UMRA variable (default: %PowershellAgentSessionId%) that used in subsequent actions that use the session.

For more information on the Powershell Agent service session, see the following topics:

*Powershell Agent service session on page 78*

*Script Action: Setup Powershell Agent service session*

*Script Action: Release Powershell Agent service session on page 606*

*Configuration section on page 27*

*Script Action: Check Powershell Agent service session*

The action is used to check if an Powershell Agent service session still exists. A number of UMRA Powershell actions use a Powershell session. With action **Setup Powershell Agent service session**, such a session is initialized at the Powershell Agent service session. When the session is idle for a configurable time interval, the session is released by the

Powershell Agent service. When the session is released, all variable objects stored with the session are destroyed. With this action, you can check if the action still exists at the Powershell Agent service. If this is not the case, the session must be recreated.

The input property **Session ID** specifies the Powershell Agent service session ID, as generated with action **Setup Powershell Agent service session**. The output property variable **Session flag** indicates if the session was found, yes or no.

With this action, you can only check Powershell Agent Service sessions that are created within the same client context: the same *UMRA session* on page 81 must be active. E.g. if the UMRA Forms client is restarted, a new UMRA session is created and old existing sessions cannot be checked.

More information:

*Powershell Agent service session on page 78*

*Script Action: Setup Powershell Agent service session*

*Script Action: Release Powershell Agent service session on page 606*

*Configuration section on page 27*

*Script Action: Release Powershell Agent service session*

The action is used to release an existing Powershell Agent service session. A number of UMRA Powershell actions use a Powershell session. This action is used to delete such a session when it is no longer needed. The action has only a single input property, **Session ID** that identifies the existing Powershell Agent service session.

For more information on the Powershell Agent service session, see the following topics:

*Powershell Agent service session on page 78*

*Script Action: Setup Powershell Agent service session*

*Script Action: Release Powershell Agent service session on page 606*

*Configuration section on page 27*

## 5. Context sensitive Help

### 5.1. UMRA PSM Domain Controllers Overview

This window gives an overview of the installation status and health of the PSM notification packages on the domain controllers, and is the starting point for managing the installation and configuration of PSM.

The PSM Notification package is custom UMRA software (UmraPsmW32.dll or UmraPsmX64.dll) that is loaded by Windows on the domain controller. The package is signalled by Windows on each password modification. The PSM Package will forward this signal to the UMRA service. The PSM package must be installed on each domain controller in the domain in order to catch all password modifications.

#### Domain Controller List

##### Domain controller

The name of the domain controller.

##### Domain

The full DNS name of the domain.

##### Version

The version of the currently installed PSM Notification Package on the Domain Controller.

##### Current Status

Shows the current installation status of the PSM package on the domain controller. see *UMRA PSM installation status* on page 126 for the possible values.

#### Small Buttons

In the top right of the window are two square buttons. Use the button with the + sign to manually add a domain controller to the list shown. Use the button with the X sign to manually remove a domain controller from the list shown. This will not install or un-install any software, only

which Domain Controllers are listed. Only needed when you have disabled the automatic detection, or to remove a DC that has been dismantled.

### **Buttons**

#### **Install/Upgrade**

Starts the installation and upgrade wizard to start installing the current version PSM Package on the (selected) domain controllers.

#### **Delete**

Un-installs the PSM package from the selected DC.

#### **Options**

Opens a dialog where you can perform miscellaneous actions on the selected DC, such as configure the behaviour of the PSM packages on the selected service, access the log file of the PSM Packages, or initiate a reboot of the selected domain controller.

#### **Advanced**

Opens a dialog where you can configure several settings regarding to the performance of this overview window, such as Automatically discovering new Domain controllers, modifying the managed domains, etc.

#### **Refresh**

Reevaluates the status information of all domain controllers in the list, and checks for newly added Domain Controllers if automatic discovery is active.

## **5.2. Installation and upgrade wizard- Installation and upgrade options**

Specify here on which domain controller(s) the PSM Package should be installed, and press **Next**.

**Install or upgrade UMRA PSM on all domain controllers in a domain (Recommended)**

Will browse for all domain controllers in a domain (specified shortly), and offer to install the package on all Domain controllers found.

#### **Install or upgrade UMRA PSM on a specific Domain Controller**

Installs the PSM package on a specific domain controller. Note that the Package must be installed on all domain controllers in a domain in order to catch all password reset events.

#### **Install or upgrade UMRA PSM on all selected Domain Controllers**

Installs the package on all domain controllers that were selected in the overview window.

### **5.3. Installation and upgrade wizard - Specify the target domain**

Specify the domain name of the domain that will be searched for Domain controllers and prepare installation, and press **Next**.

### **5.4. Installation and upgrade wizard - Specify the target domain controller**

#### **Domain Controller Name**

Specify here the name of the domain controller on which to install the UMRA PSM notification package.

### **5.5. Install/upgrade software**

A list of the domain controllers found is presented.

Wait until the Installation/progress text at the bottom reads "Waiting for user input".

Select **Install/Upgrade** to install the PSM package on each domain controller listed as "ready to install".

Next wait until the Installation/progress text at the bottom reads Installation/upgrade Completed.

Select **Close** to return to the main overview window.

## 5.6. Delete Options

### **Delete the UMRA PSM software from the selected domain controllers**

Deletes the PSM package from the domain controllers that were selected in the main list when the delete button was selected.

### **Delete the UMRA PSM software from other domain controllers**

You can specify here Domain Controllers that are not yet in the overview list of domain controllers. They will be added to the overview window. Then you select the DC's from the overview list for which you want to delete the PSM package and select the delete button again.

So, only option 1 actually deletes the software. option 2 allows to browse for more DC's, and reissue the delete request, using option 1.

## 5.7. Domain Controller Options

Here you may perform specific actions on the selected Domain Controllers

### **Configuration**

Specify here configuration settings for the working of the PSM package itself.

### **Notify UMRA service of password changes**

When selected, the PSM Notification Package on the server will connect to the UMRA service to inform the service of any password changes. This is the default setting. If not selected for all DC's in a domain, PSM will not operate properly.

When unselected, the PSM Notification Package will be switched off on the selected Domain Controllers, and NOT send any notification to the UMRA service. The PSM Package runs as it were in stand-by mode. Use this only to switch off the package in case of severe issues.

### **Enable Logging in the PSM log**

When selected, the PSM Notification package will log information of its actions in the UmraPsm.log file, located in the system32 directory of the Domain Controller. Default this logging is turned on.



**Maximum log detail level**

Specify the detail of the messages logged, when logging is enabled.

**Update**

Selecting the update button applies the configuration settings to the selected Domain Controllers.

**Status****View log file**

Select this button to open the UmraPsm.log file on the specific Domain controller, to view detailed info regarding the performance of the PSM Notification Package.

**Reboot****Reboot options**

Select this button to be able to remotely reboot the selected domain controllers.

After installation or un-installation of the PSM package, the Domain Controller(s) involved have to be rebooted in order to complete the process. It is recommended to use the standard protocol for rebooting domain controllers that applies to your organization. However, if required, it is possible to schedule a forced reboot of the selected domain controllers by selecting the button.

**5.8. Reboot options****Schedule a reboot**

Select to initiate a reboot after the selected number of minutes.

**Reboot immediately**

Select to initiate a reboot immediately.

**OK**

Initiates the selected reboot.

**Cancel**

Exit window without rebooting.

## **5.9. Refresh options**

Select here which domain controllers must be refreshed in the overview.

## **5.10. Advanced Settings - general settings**

### **General settings regarding the PSM overview**

#### **Auto Discovery Mode**

When selected, the UMRA console will automatically search for all domain controllers in the managed domains when the Overview window is opened or refreshed. The default setting is selected.

#### **Auto Refresh Mode**

When selected the status of the DC's is refreshed when the overview window is opened or modified. Otherwise the status of the DC's is only refreshed when the refresh button is pressed. Default is on.

## **5.11. Advanced Settings - domains**

Here is a list of all domains which are automatically browsed for new domain controllers when appropriate. List here all domains for which you use PSM.

Select the add button to add domains, and the remove button to remove domains.

Note that the remove button will not remove domain controllers from the overview list, nor uninstall PSM, only remove the domain from the browse list.

## **5.12. Select domain controller wizard - Specify the target domain**

Specify the domain name in order to find all Domain Controllers in the domain.

### **5.13. Select domain controller wizard - welcome**

Specify a method to determine the target domain controllers, and select **Next**.

### **5.14. Specify the name of the domain controller**

Enter the name of the specific domain controller to manage.

### **5.15. Password Synchronisation Manager service settings**

These options define what the UMRA service does when it receives a notification of a password change from a PSM package on a Domain Controller.

#### **Enable Password Synchronisation Manager**

If the enable flag is switched of, the service only will acknowledge in the log (when the appropriate loglevel is enabled), that a PSM notification has been received by the service, but no project will be executed.

#### **UMRA project to Execute**

If enabled, the specified user-provided project will be executed on a notification from the PSM package.

The project may examine the values of several predefined variables to decide on the actions to initiate.

See *UMRA PSM Script variables* on page 127 for details

### **5.16. IDD\_TAB\_ACTIONITEM\_LN\_QUERY\_ITEMS-forwarded**

### **5.17. IDD\_TAB\_ACTIONITEM\_LN\_ACL -forwarded**

forwarded to IDH\_HELP\_ACTION\_LN\_UPDATE\_ACL

**5.18. IDD\_TAB\_ACTIONITEM\_CYCOS\_GET\_ATTACHMENT****5.19. IDD\_TAB\_ACTIONITEM\_CYCOS\_GET\_CUSTOM****5.20. IDD\_TAB\_ACTIONITEM\_CYCOS\_SET\_CUSTOM****5.21. IDD\_DIALOG\_CYCOS\_CUSTOMFIELD\_OUTPUT****5.22. Value of text item.**

Specify here the value of the text item.

Value property name	Description	Typical setting	Remarks
Text	The exact text value of the resulting field.		If the general option "Merge if exist" is specified the text is appended to the existing value of the item.

**5.23. Value of text list item**

Item type: Text List

Value property name	Description	Typical setting	Remarks
Operations	Specification how to merge the new text values with the current one. Options are:  Set (unconditionally replace existing values with specified values)  Append values(s)  Insert value(s) at begin  Remove (no error if not found)  Remove (error if not found)	Set	
Text item values	A list of new text values.		

## 5.24. Value of date-time item

Item type: Date-time

Value property name	Description	Typical setting	Remarks
Date time value specification	The date-time value to set the field to		if Specified by a variable, it should be a UMRA Date-time type variable.
Date time operation	<p>Specifies how to merge the item with existing values. There are 3 options.</p> <p>1) Set item value to the specified date-time value.</p> <p>This results in a single date-time as specified</p> <p>2) Append the specified date-time value to the current values</p> <p>Any existing list of date-time values is extended with the new value.</p> <p>3) Insert the specified date-time value at the beginning of the current list of date-time values.</p> <p>Any existing list of data-time values is retained, and the new value is added in front of the current values.</p>		

## 5.25. Value of numeric item

Item type: Numeric

Value property name	Description	Typical setting	Remarks
Number value specification	The numeric value to set the field to		if specified by means of a variable, the result must be resolvable to a numeric value. If a variable is used it is therefore best to specify only a single UMRA variable of the numeric type.

Number operation	<p>Specifies how to merge the item with existing values. There are 3 options.</p> <p>1) Set item value to the specified number value. (default)</p> <p>2) Append the specified numeric value to the current values</p> <p>Any existing list of numeric values is extended with the new value.</p> <p>3) Insert the specified numeric value at the beginning of the current list of numeric values.</p> <p>Any existing list of numeric values is retained, and the new value is added in front of the current values.</p>		
------------------	---	--	--

## 5.26. Built-in variables

A built-in variable is a variable that is generated automatically by UMRA. These variables can be used for logging and script-debugging purposes. The built-in variables are predefined. The following list shows all of the built-in variables.



Variable name	Example value	Description
%NowYear%	2006	The current year (2006,...). The value is generated when a script is executed by either the UMRA Console or UMRA Service application.
%NowMonth%	11	The current month (1,...,12). The value is generated when a script is executed by either the UMRA Console or UMRA Service application.
%NowDay%	26	The current day (1,...,31). The value is generated when a script is executed by either the UMRA Console or UMRA Service application.
%NowHour%	14	The current hour (0,...,23). The value is generated when a script is executed by either the UMRA Console or UMRA Service application.
%NowMinute%	35	The current minute (0,...,59). The value is generated when a script is executed by either the UMRA Console or UMRA Service application.

%NowSecond%	9	The current second (0,...,59). The value is generated when a script is executed by either the UMRA Console or UMRA Service application.
%TimeStamp%	20080528-111433	A timestamp string representing the current date and time in the format: YYYYMMDD-HHMMSS (year month day - hour minute - second).
%CurrentSystemDate%	11:14 05/28/2008	The current date and time stored as a date-time variable type.
%UmraFormSubmitAccount%	Domain_A\WilliamsJ	The name of the user account that runs the UMRA Forms or UMRA Console application and submits a form to the UMRA Service. The variable value is determined by the UMRA Service when a form submit request is received.

%UmraFormSubmitDomain%	Domain_A	The name of the domain of the user account that runs the UMRA Forms or UMRA Console application and submits a form to the UMRA Service. The variable value is determined by the UMRA Service when a form submit request is received.
%UmraFormSubmitUsername%	WilliamsJ	The username of the account that runs the UMRA Forms or UMRA Console application and submits a form to the UMRA Service. The variable value is determined by the UMRA Service when a form submit request is received.
%UmraPath%	C:\Prgram Files\UmraService	The program directory of the UMRA Service of UMRA Console application.
%SystemRoot%	C:\WINDOWS\system32	The path of the system32 folder of the computer that runs the UMRA application.
%UmraProjectName%	CreateUser	The name of the UMRA project that is currently executed.

%UmraProjectNameStack%	CreateUser GetAccountNames	A text list variable containing all projects that are currently being executed up to the current project. The variables gets multiple values when using for-each constructions and when projects execute other projects.
%UmraClientComputerName%	HERMES	The name of the client computer from which the form is submitted. This variable is only available when using the UMRA Forms client application in combination with the UMRA Service. The value is generated when a submit button is pressed in a form.

Note: When the value of a built-in variable is generated, an existing value of a variable with the same name is overwritten.

## 5.27. Condition criteria - Setup

Using the **IF-Then-Else** script action you can evaluate a condition. A condition is always setup using the following components:

### IF

The IF component includes one or more criteria for the condition.

**Then**

In the **Then** section you specify which action should be executed if the condition is met

**Else**

In the **Else** section you specify which action (if any) needs to be executed if the condition is not met.

**5.28. Condition criteria - Setup criterion**

In the **Setup criterion** dialog box you must specify how the variable in your If-Then-Else condition should be evaluated. The variable type can be text, numeric, date-time or a boolean.

The equation operator can be one of the following:

- has no value or does not exist
- equals (case sensitive and case insensitive)
- contains (case sensitive and case insensitive)
- starts with (case sensitive and case insensitive)
- ends with (case sensitive and case insensitive)

If you need to obtain an inverted result (e.g. does NOT equal), the option Invert the result should be marked.

**5.29. Configure predefined variables**

The script of a User Management Resource Administrator project can contain variables. To run a project, the variables need to be configured. The project can contain additional information for the variables. With the option **Configure predefined variables**, this information is shown for each variable to help you specify the variable values.



The top section of the window describes the variable. It shows the **variable name** (%Domain%), a **description** and some **example values**. In

the bottom section of the window, you need to specify a variable. If the value of the variable should be **the same each time the script is executed for a line of the input data**, then select the option **Constant** and specify the value of the variable. If the value of the variable corresponds with an input data column, then select the option **Input data column** and select the column from the list with available columns.

**See also:**

Help on help

*UMRA Basics* on page 3

*Managing script actions* on page 712

### 5.30. Control running UMRA service projects

In this window are listed the UMRA service projects of which the scripts are currently actively running on the UMRA service. This window is mainly informational.

The **Abort** button can be used in case of urgent need to abort a specific script.

Under normal conditions this option should not be used. However, in some circumstances it may be required to abort a specific script. For instance in order to stop a lengthy script that contains an error.

### 5.31. Data specification - Text list

User Management Resource Administrator can process various different data formats: text, numbers, date and time values etc. In specific cases, the data can not be specified as a single value, but as a series of values. For instance, the *Script Action: Set User Global Group Memberships* [http://www.tools4ever.com/resources/manual/usermanagement6/script\\_action\\_set\\_user\\_global\\_group\\_memberships.htm](http://www.tools4ever.com/resources/manual/usermanagement6/script_action_set_user_global_group_memberships.htm) supports the property **Global groups**. This property holds a series of global group names. In other words, this single property specifies

multiple names. If you want to assign global group memberships using variables you want the variable to hold multiple values. For example:

1. Variable **%GroupSet%** contains the groups **GlobalGroupA**, **GlobalGroupB**, **GlobalGroupC**
2. In the script action **Set User Global Group Memberships**, the property **Global groups** is set to **%GroupSet%**

To support this mechanism, you can set the value of a variable using the multivalue type **Text list**. To start, add an action *Script Action: Set Variable* on page 544 to the script. Edit the properties of the action and select **Text list** as the type of the variable value.



Next, click the **Browse** button. The **Specify input** window is presented. The window shows a list with all the current values of the variable. Click the **Add**, **Edit** and **Delete** buttons to manage the values of the variable.



When you click the **Add** button, you can directly specify a new value for the variable or you can search the network to find one or more items.



If you select the option **Specify name**, you can simply enter the value and click **OK**. If you select the option **Specify search method**, you can specify the type of search and the format of the output names. Once the search method and name format are selected, click **OK** to continue. When ready, the results will be shown in the **Specify input** window.

**See also:**

*UMRA Basics* on page 3

*Script Action: Set Variable* on page 544

*Script Action: Map variable* on page 567

*Script Action: Set User Global Group Memberships* on page 88

## 5.32. Database query - Database specification

In the **Database type** list box, you can select one of the following database types:

### 1. MS Access (Jet) databases

Allows you to connect to an MS Access database through the Jet engine. Jet is the Database Management System (DBMS) which underlies MS Access and also Visual Basic, as well as MS Word and MS Excel.

#### Next steps:

*Database setup - MS-Access (Jet) on page 627*

*Database query - Query on page 626*

### 2. All other databases

Choose this option if you wish to connect to any other (relational) databases (SQL server, Oracle, etc.) through a *connection string* on page 628. A connection string includes the source database name and other parameters needed for the initial connection. The default value is an empty string.

#### Next steps:

*Database setup - Other databases on page 628*

*Database query - Query on page 626*

## 5.33. Database query - Query

#### Previous steps:

1. *Specifying the data source on page 626*
2. *Specifying the MS Access database on page 627*



Once you have completed these steps, you are ready to create a database query. For example, suppose we have an MS Access table called Users.MDB which contains a table **Users** with the following information:



The query syntax for retrieving all the table data is:

**SELECT \* FROM USERS**

where "USERS" is a table in the **Users.mdb** database file.

Since the query can be considerably more complex than the one for this example, we would advise you to construct the query in MS Access until you are absolutely certain that your query returns the required result. If this is the case, then copy the SQL query into UMRA as follows:

1. Select the **Query** tab in the **Setup generic table** dialog box
2. Paste the query into the **Database query** window.



3. Click the **Run test** tab and click the **Test** button to check once more if the query is working correctly.



## 5.34. Database setup - MS-Access (Jet)

### Specifying an MS Access database

1. Drag the **Table** form field from the **Actions-Network-Form field** window to the Form window of your project.
2. Select the option **Generic table** and click the **Configure** button. The **Configure Table** dialog box will appear.
3. Click the **Configure** button and select the option **Database query** from the **Table type** list and click **OK**. The **Setup Generic table** dialog box will appear.



4. Click the **Configure** button in the **Database specification** window to specify the MS Access database (.mdb file) you wish to use (in the screenshot below a connection is made to the database Departments.mdb).



Note: Please ensure that the specified path to the database can be accessed by the UMRA module. In most cases you will define a share for storing the MS Access databases (e.g. \\<Computer name>\<Share name> instead of pointing to an absolute path name).

### 5.35. Database setup - Other databases

In UMRA, you can connect to a wide variety of databases (SQL server, Oracle, etc.) using an OLE DB connection.

#### Establishing a connection

To establish a connection, click the **Configure** button in the **Configure database connection** section and select the OLE DB connection you wish to establish. You will need to provide details regarding the source database name and other parameters.

Once you have established a successful connection, the connection string will be displayed in the **Database connection string** window. A connection string includes the source database name and other parameters needed for the initial connection. The default value is an empty string.



#### Changing the connection string

In some cases you may wish to customize this connection string (e.g. when there is no OLE DB provider for your specific database or if you wish to add certain keyword values for an existing connection. In the

table below you will find a list of valid names for keyword values within the `ConnectionString`.

Name	Default	Description
Application Name		The name of the application, or '.Net SqlClient Data Provider' if no application name is provided.
AttachDBFilename -or- extended properties -or- Initial File Name		The name of the primary file, including the full path name, of an attachable database.  The database name must be specified with the keyword 'database'.
Connect Timeout -or- Connection Timeout	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
Current Language		The SQL Server Language record name.
Data Source -or- Server -or- Address -or- Addr -or- Network Address		The name or network address of the instance of SQL Server to which to connect.
Encrypt	'false'	When true, SQL Server uses SSL encryption for all data sent between the client and server if the server has a certificate installed. Recognized values are true, false, yes, and no.

Initial Catalog -or- Database		The name of the database.
Integrated Security -or- Trusted_Connection	'false'	When false, User ID and Password are specified in the connection. When true, the current Windows account credentials are used for authentication.  Recognized values are true, false, yes, no, and sspi (strongly recommended), which is equivalent to true.
Network Library -or- Net	'dbmssocn'	The network library used to establish a connection to an instance of SQL Server. Supported values include dbnmpntw (Named Pipes), dbmsrpcn (Multiprotocol), dbmsadsn (Apple Talk), dbmsgnet (VIA), dbmslpcn (Shared Memory) and dbmsspxn (IPX/SPX), and dbmssocn (TCP/IP).  The corresponding network DLL must be installed on the system to which you connect. If you do not specify a network and you use a local server (for example, "." or "(local)"), shared memory is used.
Packet Size	8192	Size in bytes of the network packets used to communicate with an instance of SQL Server.
Password -or- Pwd		The password for the SQL Server account logging on (Not recommended. To maintain a high level of security, it is strongly recommended that you use the Integrated Security or Trusted_Connection keyword instead.).

Persist Security Info	'false'	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.
User ID		The SQL Server login account (Not recommended. To maintain a high level of security, it is strongly recommended that you use the Integrated Security or Trusted_Connection keyword instead.).
Workstation ID	local computer name	The name of the workstation connecting to SQL Server.

For more information please check the *Microsoft website*  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemDataSqlClientSqlConnectionClassConnectionStringTopic.asp>.

### 5.36. Expiration date

Specify here the exact date and time after which the object or user will expire.

If a variable is used to specify the expiration date, it must be an UMRA date-time variable, and not a text variable.

The exact interpretation of the expiration date depends on the particular script action:

**Script actions** *Script Action: Create User (AD) on page 3, User - Edit user (AD) on page 37 etc*

The specified user account will be expired at the first day following the specified date-time.

**Script action** *Lotus Notes - Get certifier on page 392*

Input property. Any certificates generated with the resulting certifier object variable will expire at the specified time. If this property is not specified, an expiration date of 2 years from the current time is used.

If the certifier is used to register a person, this date consequently specifies the expiration date of the user account.

## **5.37. For each - Input variables**

### **Pass variable method specification**

These options specify which variables that are available in the current script are passed on to the project that is called repeatedly.

Passing on values of variables to the called project can cause considerable overhead in specific circumstances, especially when some variables contain a lot of data, such as an table with many entries. Therefore these options let you control which variables to pass, and which not, in order to enhance performance.

### **Pass all variables**

All variables that are available in the current script are passed on to the repeatedly called project. This means that all variables that are defined in the current script are available for the called project. Use this if you do not know which variables you will actually use in the called script, or if they do not contain large amounts of data.

This is the default setting.

**Pass "for each" variables only**

Only the variables that are bound to a specific column in the table over which the "for each" is performed, as specified on the "fore each" tab, are passed on.

This gives the best performance. Use this if you do not want to make available any other variables from the current script in the called project.

**Pass "for each" variables and the specified variables**

In addition to the "for each" variables, the variables specified in the specified Variable list are also passed on to the called script. Use this option if you want to use some specific additional variables from the current script in the called script.

**Pass all but the specified variables**

All variables are passed to the called script, except those specified in the Variable list. Use this if you want access to most variables in the called project, but there are some large variables you do not need in the called project.

A variable that often does not need to be passed to the called project is the "table" variable, that was used to generate the "for each" variables. As the called project usually does not need access to the entire table, but only to the generated variables that represent the current row in the table, the original "table" variable can be added to the list of excluded variables.

**Variable list**

This list contains the variables that are explicitly passed on when the option '**Pass "for each" variables and the specified variables**' is chosen

The list contains the variables that are explicitly not passed on when the option '**Pass all but the specified variables**' is chosen.

It is not used with the other options.

### 5.38. Form action - Check form input

The **Check form input** function is used to check the input of the submitted form. If the input is not correct, a message is shown to the end user of the form. This action is always executed as the first action when a form is submitted. The following options are available in this window:

Form variables check specification

Form objects such as input boxes and table entries can be assigned to a variable. You can either check the content of these variables (e.g. to make sure that the user has selected a table entry before clicking an action button) or perform a length check on the entered text (e.g. to make sure that a password has the required length or that a phone number has 10 digits, etc.). If a check is specified, the check status will appear in the **Checking** column.

#### Variable contents check

When the user selects a table entry or enters text in an input box and then clicks the form button, the UMRA Service will process the submitted form. With the **Check form input** action, the UMRA service can check the specified variables. In the screen below, the variable %UserName% should contain the value of a selected user account. By checking the contents of this variable, you can check if the user has actually selected a table entry before clicking the action button. If not, an error message can be displayed to inform the user.



#### Message text shown when variable equals check value

The contents of the message shown to the end-user:



#### Variable length check



You can also check if the string which the user has entered, is of the required length (see example below) by entering a minimum and / or maximum length. An error message can be displayed when the string length does not meet the specified criteria.



### 5.39. Form action - Execute script of form

A form project consists of a form and a script. In a form project, this script is executed when the end user clicks a form button. With this action, you specify that the script of the current form project should be executed. This action has no additional parameters to be configured.

When the script is executed, you can show a message box to the end-user of the UMRA forms application. This is done by using the variable **%ScriptMessage%**. If a variable with the name **%ScriptMessage%** is found, a message is shown to the end-user.

For general information on project scripts, forms and project script execution, see *UMRA Basics* on page 3.

### 5.40. Form action - General

When a form is submitted, it is sent to the UMRA service for further processing, for example: the script of the form project is executed. In UMRA, a number of form actions exist. A form action is an action that is executed by the UMRA service (or UMRA console application for local form projects) as part of the processing of a form that was submitted. Form actions can be associated with various form fields:

1. **Button form field:** For each action button, a number of form actions can be specified.
2. **Checkbox form field:** For a checkbox you can define form actions for both the checked and unchecked state.

When an action button is pressed, the form is submitted. The contents of various form fields is stored in variables and sent to the UMRA

service. Note that the form actions are not directly related to the form. Instead, form actions are defined for form fields. For different submit buttons (example: disable account, enable account, unlock account) you can (and should) define different form actions.

**Important:** Form actions are very different from script actions. Script actions are part of a project script. An example of a script action is the creation of an user account in Active Directory. See *Manage script actions* on page 712 for more information on script actions. A form action is an action executed by the UMRA service as a result of an end-user submitting a form.

#### Setting up the form actions for an action button

1. Doubleclick the button form field
2. Click the **Manage actions** button in the **Button actions** section.

The **Configure form actions** window is shown:



The **Form actions** window shows all the form actions currently defined for the form field. Use the **Add**, **Edit** and **Delete** buttons to configure the form actions. Use the up and down keys to change the order of the form actions.

To setup the form actions for a **Checkbox** form field, the procedure is very similar. Normally, you only want to configure the **Set variable** action for a checkbox form field.

### 5.41. Form action - Iteratively execute project script

This form action is used to execute the script of the project multiple times when the form is submitted. The action is used for *multiple-select tables* on page 659 and the script is executed **for each selected table items**.

Example: suppose a form contains a table with user accounts. In the table, multiple user accounts can be selected. When the submit button is clicked, a user account property must be updated for each of the selected user accounts. In such a scenario, this action is used.

The iteration of a script execution is controlled by a form field, normally a multiple-select table. The variables associated with the specified form

field contain multi-values when the form is submitted. The script is executed for each of the values of the variable(s).

**Example:**

In a form, a multiple select table with user accounts is shown. The account names are stored in variable %UserName%. In the script a variable %Domain% is set to DOMAIN\_A. The user selects 2 account names (John and Fred) and presses the submit button.

Now the script is executed twice with the following variable settings:

Iteration 1:

%Domain%=DOMAIN\_A

%UserName%=John

Iteration 2:

%Domain%=DOMAIN\_A

%UserName%=Fred

**See also:**

*Table form field - Options* on page 659

## 5.42. Form action - Return current form

This form action is always executed as the last action. With this action the same form is returned to the application that submitted the form. With this form you can also configure how the different form fields must be returned: with their current values or in the original state.

In the **Specify form fields restore options** window, all form fields with a name are shown. By checking a form field, the current state of the form field is restored when the form is shown.

There are two options for the way in which the current form should be returned:

1. **Input restored:** the selected item in the table is still selected. In this case, you do not need to select the item from the table. Instead, it remains selected.

2. **Re-initialized:** The table is shown as if the form was presented for the first time: no items are selected.

To specify the name of a form field, see *Form fields - Name* on page 646

### 5.43. Form action - Return other form

This form action is always executed as the last action. With this action a form is returned to the application that submitted the form.



This action can be used to setup a wizard with UMRA Forms. As response to a form submit button (e.g. **Next**), a script is executed and the next form is returned. A variable can be specified for the next form.

#### Form project name

The name of the form project that must be returned.

#### Reset variables

By default, all variables that exist when the project script is executed are returned. The values of these variables can be used in the returned form. You can control this with *Script Action: Delete variable* on page 562. To reset all variables, select this option.

### 5.44. Form action - Set variable value

With this action, you can setup the value of a variable. In most cases this action is used for a checkbox or when a form has multiple submit buttons. To configure this action you need to specify the name and the value of the variable.

### 5.45. Form action - Execute command line at client workstation

With this form action, a command line can be executed by the UMRA Forms client application. The command line is executed when a form is submitted and the action is configured for the button that was clicked to

submit the form. When the form is submitted, the command line is prepared by the UMRA Service and send back to the UMRA Forms client. If the command line specification contains variables, these will be resolved by the UMRA Service. When the UMRA Forms client processes the returned data, as the last action, it will execute the command line. When the command line is started, the UMRA Forms client continues to run.

Note: when multiple 'Execute command line at client workstation' actions are configured for a single button, only the last action will be executed. The other actions are ignored.

## 5.46. Form fields - Button

The **Button** form field is used to let the user submit or reset a form. In the screenshot below the last field, **Unlock account**, is a button form field.



Common name	Username
Aglaia Deione	deionea
Allams Limedec	limedeca
Astra Perimen	perimena
Barmors Manthyl	manthylb
Barritt Amphe	ampheb

Unlock account

To add a button form field:

1. Activate the **Actions-Network-Form fields** window and drag the **Button** form field to the **Form** tab of your project. This will activate the **Form** window.
2. Drop the form field at the required location. The **Configure form field** dialog box will appear:

**Button type - Reset button**

Select this option to make the button a reset button. When a reset button is clicked, the form is re-initialized, e.g. all fields of the form are reset to their original values. No other actions are executed.

**Button type - Action button**

Select this option to make the button an action button. With an action button, the form is submitted and sent to the UMRA service. The contents of the form fields are stored as variables values to be used in scripts by the UMRA service. A number of form actions can be executed by pressing an action button.

Examples: run script of the form, set variable value. etc.

**Button type - Manage actions...**

Click this button to configure the actions for the button. Note that different buttons can have different actions.

See *Form action - General* on page 635 for more information on form actions.

**Appearance - Button text**

The text displayed on the button.

**Fixed button width of ... pixels**

When selected, the width of the button is fixed and specified as a number of pixels.

## 5.47. Form fields - Checkbox

The **Checkbox form field** is used to enable or disable a specific feature in a form. In the screenshot below, a **checkbox form field** is used for option **Password never expires**.



With a **checkbox form field**, you can associate different form actions to be executed with different checkbox states. In other words, the action

will depend on whether the checkbox has been checked or unchecked. In most cases, the *Set variable* on page 638 action is executed when the checkbox is in a checked state.

#### Adding a checkbox form field

1. Activate the **Actions-Network-Form fields** window and drag the **Checkbox** form field to the **Form** tab of your project. This will activate the **Form** window.
2. Drop the form field at the required location. The **Configure form field** dialog box will appear:



#### Appearance - Text

Enter the text shown next to checkbox.

#### Initial state - Not checked

Select this option if the checkbox must have an unchecked state when the form is loaded:



#### Initial state - Checked

Select this option if the checkbox must have a checked state when the form is loaded:



#### Determined by variable

Select this option if you want the checkbox state to be determined by the value of a variable (e.g. to show for a selected user whether the user should change his password or not).

Unlike the options **Initial state - Not checked**, and **Initial state - checked**, where the administrator predetermines the initial state of a checkbox, it is also possible to use the value of a variable to set the initial state of a checkbox.

#### Example

Suppose you want to create a two-step Wizard for resetting passwords. In step 1, the user is selected from a generic table and in step 2 the **Password never expires** checkbox needs to be displayed. The state of

this checkbox should reflect the **Password never expires** setting for the selected user in Active Directory.

In this case, you would need to create two forms. The first form will contain a generic table with user data, including the values for the attribute **userAccountControl** "Password never expires" which is either "Yes" or "No". This value is then stored in the variable **%NeverExpired%**. In the second form, a checkbox is inserted with the following properties:



If the variable **%NeverExpires%** holds the value "Yes" for the selected user, the initial state of the checkbox will be checked. If **%NeverExpires%** is "No", the initial state of the checkbox will be unchecked.

Note - the example given above does not work for all the userAccountControl user flags. Property flags such as PASSWORD\_EXPIRED and DONT\_EXPIRE\_PASSWORD cannot be retrieved using an LDAP query (it is possible to set the values for these attributes, but not to retrieve them).

#### **Configure actions when form is submitted - Actions when checked**

Click the **Configure** button to setup the actions that must be executed when the form is submitted and the checkbox is checked. See *Form action - General* on page 635 for more information on form actions.

#### **Configure actions when form is submitted - Actions when unchecked**

Press the **Configure** button to setup the actions that must be executed when the form is submitted and the checkbox is unchecked. See *Form action - General* on page 635 for more information on form actions.

## **5.48. Form fields - Display**

When designing a form, you can configure a number of display characters for each form field. These display characteristics determine how the form fields are presented on the form. With the display characteristics, you can also configure the position of form fields relative to each other.



**Configuring the display characteristics**

1. Double click the form field element or select the form field element and press Enter.
2. Select the **Display** tab. The **Configure form field** dialog box will appear:

**Horizontal alignment**

With this option you specify the horizontal alignment of the form field. If the width of the form field does not exceed the width of the area used to draw the field in the form, this specification has no meaning (for instance for an picture form field, or vertical space form field).

**Font style**

The font used to draw the text of a form field. For each form field, you can use a different font. For more information on fonts in forms, see [Form properties - Fonts](#).

**Left margin**

Specify the left margin of the form field in percentages of the total form. The left margin is used to shift form fields to the right on a form. By default, form fields are drawn below each other with a *fixed left margin* on page 768. By specifying a non-zero left margin, the form field shifts to the right. If the cursor position is not increased when the previous form field was drawn, the left margin is relative to the right side of the previously drawn form field. See the option **Position control** further in this topic. Since the left margin is specified as a percentage of the total form width, the actual margin varies if the size of the form window changes.

**Field width - Limit width to ... % of form**

By default, a form field can use all horizontal space from the current horizontal position to the right margin of the form window. By limiting the width of the form field, you can control the area of the form used to draw the form field.

**Vertical offset**

To align form fields better, you can shift individual form fields in vertical direction by specifying this field.

**Position control - Move cursor to next line for next field**

This is an important field, used to place form fields next to each other.

By default, form fields are drawn below each other with a *fixed left margin* on page 768. If you unselect this option, the current form field is drawn and the next form field will be drawn next to it. If you select this option, you must specify the option **Field width** and limit the width of the current form field.

**Tab control - Activated when pressing tab characters**

In a form with multiple fields, the focus jumps to the next field if you press the TAB character. If you press the TAB key with the SHIFT key pressed, the focus moves back to the previous form field. By checking this option, the current form field becomes part of the loop. If the option is unchecked, pressing the TAB character cannot bring the focus to this form field.

**Text - foreground color**

The specification of the foreground color, usually the text of a form field. Click the **Edit** button to change the current color. By default, the foreground color is black.

**Background color**

The specification of the background color. Click the **Edit** button to change the current color. By default, the background color is white.

## 5.49. Form fields - Input text

The **Input text form field** is used to let the user of the form specify a text value. Examples: first, middle and last name, password, phone number, description of a user account, extra SMTP E- mail address. In the screenshot below the fields next to the text **New password** and **Confirm password** are input text fields.

**Adding an input text field**

1. Activate the **Actions-Network-Form fields** window and drag the **Input text field** form field to the **Form** tab of your project. This will activate the **Form** window.

2. Drop the form field at the required location. The **Configure form field** dialog box will appear:

**Text**

Specify the text that is initially displayed in the input text field. When you tab through the form and the form field gets the focus, all of the content of the input field is selected. When you start typing the input text, the selection is removed.

**Variables**

Select a variable from the list and click the **Insert** button to insert the variable name at the current position in the **Text** field. At runtime, when the form is shown, the value of the variable is shown.

**Text field support multiple lines with ... visible lines**

Select this option to make the text field a multi-line input field. In this case, the form user can specify a number of input lines for the input field. When the user enters the text, the form field will wrap to the next line automatically. If you do not select this option, the field height is limited to a single line of text.

**Margin between field border and text of ... pixels**

The specified margin is used to draw a border around the input text field when specified.

**Password style, all characters shown as an asterisk (\*)**

When the user enters text in the input text field, each character is represented as an \*-character. This style is normally used to specify passwords.

**Draw border of input field**

Draw a border around the input text field. The border clearly indicates the position of the input text field.

**Accept carriage return (<Enter>) characters**

When selected, the text in the input field can be entered using <Enter> characters. Such a character moves the cursor to the next line in a multi-line input text field.

**Variable - On submit, store contents in variable**

Specify the name of the variable that is used to pass information entered in the input text field, to for instance the script of the form project. When a submit button is pressed, the entered text is stored as the value of the specified variable. If you do not

specify a variable name in this field, the input text field cannot be stored. The list shows the names of the variables found in the various project components. Instead of selecting a variable from the list, you can also simply enter the name of a (new) variable.

### 5.50. Form fields - Name

Each form field can have a name. The name is used to refer to the form field. The name of a form field is optional if no other item refers to the form field. If a form field is referred to, the form field must have a name. The form field name can be any text. It is recommended to use a short descriptive name for a form field.

#### Specifying the name of a form field

1. Select a form field from the list of form fields.
2. Right click the mouse and select the menu option **Edit form field**.
3. Select the **Name** tab.
4. Specify the name and click **Apply** or **OK**. The name should be unique within the form.

### 5.51. Form fields - Picture

The **Picture form field** is used to clarify a form, design the form according to your company standards and make a form more easy to use. A form can contain multiple pictures of any size. The most common image standards, e.g. jpg, gif, and bmp are supported. At design time, the pictures are selected from image files and then embedded into the form.

#### Adding a picture form field

1. Activate the **Actions-Network-Form fields** window and drag the **Picture** form field to the **Form** tab of your project. This will activate the **Form** window.
2. Drop the form field at the required location. The **Configure form field** dialog box will appear:



**Image file name**

The original name of the file that contains the image. An image is selected by specifying the file that stores the image. Click the browse (...) button to select an image file. Once the picture field is created, the file name no longer has a meaning: The image itself is embedded into the form. Once a form is designed, you can even delete the image files of corresponding picture without changing the form.

**Scale (form)**

You can scale the image with respect to its size in the form. A scale factor of 1.0 does not change the size of the image in the form relative to the original image size. A factor of 2 enlarges the image both in the horizontal and vertical direction.

**Preview (scaled to fit)**

Once an image file name is selected, the preview shows the image. If the total image does not fit into the window, it is scaled to make it fit. Note that in the form, the image is scaled by specifying the scale.

## 5.52. Form fields - Radio button

The radio button form field is typically used to present the user with various options in a form. In UMRA, you can either present the user with a static list of options and assign the selected choice to a variable, or you can set the initial state of a radio button using a variable.

**Creating static Radio buttons**

1. Enter a name for the radio button variable in the **Radio button variable** field (e.g. %SelectedRadioButton%).
2. Create the radio buttons you wish to display. Click the **Add** button in the **Radio buttons** section . The following dialog box will appear:



In the **Display text** field, you can enter the text for the radio button.

In the **Variable value field**, you can either enter a fixed value to be associated with this radio button or a variable (e.g. %DeleteDirectory%). When the user selects a radio button, the value associated with the

selected radio button will be stored in the radio button variable you specified in step 1 (%SelectedRadioButton%.)

In other words, if the user selects the option **Delete directory** in the example given above, the variable %SelectedRadioButton% will be set to "0" or "%DeleteDirectory%" .

### Creating dynamic Radio buttons

It is also possible to determine the initial state of a radio button by a variable value. Suppose you have the following three radio buttons:

"Delete directory", variable value "0"

"Move directory", variable value "1"

"Copy directory", variable value "2"



Depending on the value of a variable (e.g. %Check%), the initial state of a variable can be set. If the value of %Check% is 0, the first option will be set, if the value equals "1" the second option will be set, and so on.



### See also:

*Button form field* on page 639

*Checkbox form field* on page 640

## 5.53. Form fields - Static text

A static text form field is used to explain the form, form fields and form usage. In the form shown below, the text Unlock account is a static text field.



**Adding a static text form field**

1. Activate the **Actions-Network-Form fields** window and drag the **Static text field** form field to the **Form** tab of your project. This will activate the **Form** window.
2. Drop the form field at the required location. The **Configure form field** dialog box will appear:

**Text**

Specify any text you would like to show in the form in this field.

According to the display settings of the form field, the text will be shown automatically on multiple lines.

**Variables**

Enter any variable name (examples: %Domain%, %CallThisNumber%) in the **Text** field. When the form is shown, these variables will be replaced with their actual values (if no variable with the name exists, the name of the variable is shown).

**See also:**

*UMRA Basics* on page 3

## 5.54. Form fields - Table - Columns

In this window, you can configure the table columns for the form table:

- Specifying the columns to be included in the form table
- Linking columns to variables
- Sort order, positioning and display

**Specifying the columns to be included in the form table****Available columns**

Shows the available columns for the specified table. The list with available columns is shown in the **Available columns** section.

**Current column configuration**

This is the list of columns which will be displayed in the form table. Use the --> and <-- buttons to add and remove columns from **Available columns** to the **Current column configuration** list. In the form table, these columns will be displayed from left to right.

When you select a column in the list **Current column configuration**, the controls in the bottom of the window show detailed information for the selected item. At the bottom section, Column specification (selected column) you can setup the selected column.

#### Name

The name of the column. The name is predefined and cannot be changed.

#### Variable

The name of the variable linked to this column.

At runtime, this variable will be filled with the value of the selected table entry. In the example shown below for instance, the variable %UserName% has been linked to the **Username** column. The selected entry, limedeca, will be stored in the %UserName% variable and submitted to the **UMRA Service** when the submit button **Unlock account** is clicked.



Common name	Username
Aglaia Deione	deionea
Allams Limedec	limedeca
Astra Perimen	perimena
Barmors Manthyl	manthylb
Barritt Amphe	ampheb

Unlock account

#### Column width

The width of the column, specified as an percentage of the total column width. Note that you can specify a column with zero width if you want to link a column to a variable without showing the column to the end user.



**Sort on this column**

You can sort on each column in a network table and the sorting can be based on multiple columns. Normally the specification of a 1st criterion only is sufficient. Depending on the interface that is used to show the form, the end-user can change the sorting when the table is shown. Note that you can sort only on columns that are actually shown in the form.

**Sort direction**

Specifies the sort direction of the selected column.

**Alignment**

Specifies the horizontal alignment of a table column item.

**See also:**

*UMRA tables* on page 9

**5.55. Form fields - Vertical space**

By default, UMRA places all form fields just below each other with no margin. In most cases, you probably want to separate the fields in a vertical direction. With the **Vertical space** form field you can create some vertical spacing between form fields. Specify the vertical space as the number of pixels. A default value of 10 pixels is used.

For more options on formatting the display of a form, see *Form fields - Display* on page 642.

**5.56. Form fields - Table - Data refresh**

To improve performance and limit network traffic, the actual contents of a network data table are stored by the UMRA service. To do this, the UMRA service uses a local internal database in RAM that is completely self managed. The first time the information is collected, the database is empty and it might take some time to collect the data and present the form. Typically, this can take from 1 to 30 seconds. Once the data have been collected and stored, the response is much faster. The data of the internal database is shared by all forms.

Because of this mechanism, the data can be out of date. Therefore, you can specify a maximum age of the data: the **network data refresh period**. When the form is generated, the database is checked for its contents. If it contains the network data, and that data is not older than the specified number of seconds, the data is loaded from the database. In all other cases, the data is collected from the network and the database is updated.

The default value used for the **network data refresh period** is 900 seconds (15 minutes). If you specify a value of 0 seconds, the data is always collected from the network. This makes the data very accurate but increases network traffic and slows down performance.

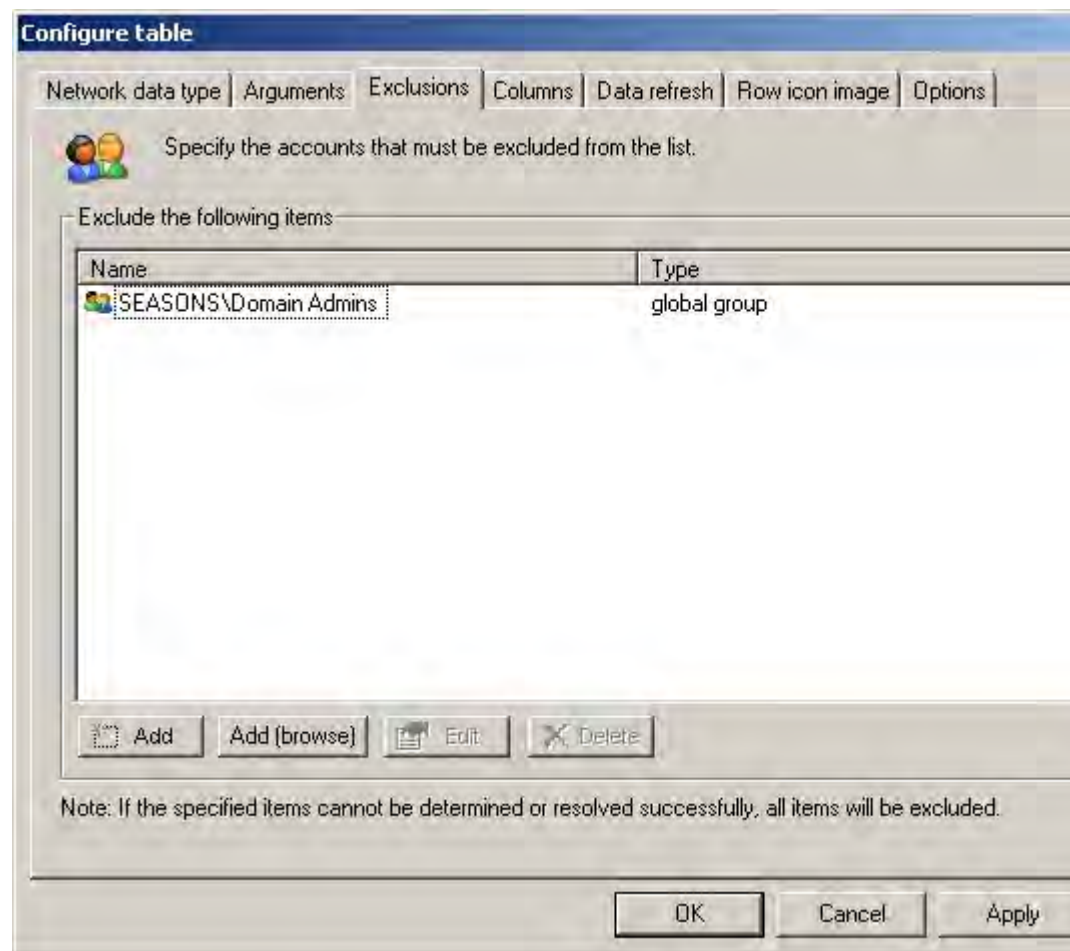
Note that user management actions executed by the UMRA service itself, will update the internal database automatically.

Example: if you have a form to delete user accounts from an OU, the users of the OU are probably shown in a table. The contents of this table are stored in the internal database. Now, if you select a user from the table, and the user is removed by the UMRA service, the data of the internal database is updated and no longer shown in the form table.

You can configure the **network data refresh period** for each form table individually. To do so, start the **Configure table** window. See *Table form field - Type* on page 660 for more information. Once a *Network data type* on page 657 has been selected, select the tab **Data refresh** and follow the instructions.

### 5.57. Form fields - Table - Exclusions

In some circumstances you might want to exclude certain accounts from a network table. In a Windows NT4 environment for instance, a table contains all users of a domain. The table is used in a form to reset passwords. In this case, you probably want to exclude the administrator accounts from the table.



Use the **Add**, **Add (browse)**, **Edit** and **Delete** buttons to configure the list with items that must be excluded.

At this moment, you can exclude members of one or more global groups. At runtime, UMRA resolves the items of the exclusion list. If an error occurs, all items will be excluded.

Example: in the example shown above, the members of group SEASONS\Domain Admins are determined when the content of the table is setup. If this fails for whatever reason, the table will be empty.

## 5.58. Form fields - Table - Fixed data

A fixed data table always has the same contents. The contents is determined at design time. This type of table is used to let an end-user select a class, division, department, OU, domain etc. in a form. The selected item of the table is stored in a variable when the form is submitted. This variable can be used in a UMRA script. By using the script action *Script Action: Map variable* on page 567 the variable can be used to determine the value of other variables. A fixed data table has one column only. The number of items (rows) is not limited. To setup the height of the table as shown in the form, see *Table form field - Options* on page 659.

To start setting up a fixed table or editing an existing table, see *Table form field - Type* on page 660.



### Table data

The contents of the fixed data table. The field shows the current contents of the fixed data table.

### Add

Click the **Add** button to add new items to the table. Note that you can specify multiple items with one add operation.

### Edit

Edits the currently selected item of the table data. A single item must be selected to activate this button.

### Delete

Deletes all of the selected entries from the table.

### Import

Imports the contents of a text file to the table. Each line of the text file is imported as a new item in the table.

### Variable

Specifies the name of the variable. This variable can be used to pass the selected item to the project script. At run-time, when the form is submitted, the selected table item is determined. If an item is selected,

the item is stored in the value of the specified variable. The variable can be used in subsequent actions performed by UMRA. If you do not specify a variable name in this field, the table selection can not be used in any subsequent action. You can select a variable from the list, or simply enter the name of a (new) variable.

**Sort table contents**

When checked, the table contents is sorted in ascending order when shown in the form. If not checked, the table contents is shown in the form as entered in the field **Table data**. In this case, you can use the up and down arrows to setup the order of table items.

## 5.59. Form fields - Table - Generic table

A generic table can contain data from a file, database query, network query or fixed data.

By including a generic table in a form, data from these sources can be displayed to the end user. Selected form data can then be used as input for an UMRA project script.

**See also:**

*Form fields - Table - Fixed data on page 654*

*Form fields - Table - Network table on page 657*

*UMRA Basics on page 3*

*UMRA tables on page 9*

## 5.60. Form fields - Table - Network call parameters

The **network arguments** complete the specification of the network data table. The network call parameters depend on the specified *network data type* on page 657. If you change the network data type, you must also change the the network arguments. In the example shown below, the specification is as follows:

1. Network data type: User accounts of an organizational unit (OU)

## 2. Network arguments:

LDAP://SPRING/OU=USA,OU=Sales,DC=seasons,DC=tools4ever,DC=local and

LDAP://SPRING/OU=Schools,DC=seasons,DC=tools4ever,DC=local+

The network data table shows the user accounts obtained from the specified OU's.



For each network data type, the syntax of the network call parameter is different. To specify the network call parameters, start the **Configure table** window. See *Table form field - Type* on page 660 for more information. Once a *Network data type* on page 657 is selected, select the **Arguments** tab. The following window is presented:



### Network data collection parameters

The list shows the arguments used to collect the network data. For each type of network data, you can specify multiple entries. The results of all entries will be presented in the network data table. If an argument can have child objects (for instance an OU having child OU's), you need to specify a + sign at the end of the argument parameter specification to include the items from the child objects as well. If different arguments share resulting items, these will be filtered out automatically. Note that each entry can contain *variables* on page 770. In that case, the exact network data that must be collected is determined when the form is generated.

### Add, Add (browse), Edit, Delete

Use these buttons to manage the contents of the list with Network data collection parameters. Note that each entry can contain variables. In that case, the exact network data that must be collected is determined when the form is generated.

### Network data type

The type of network data for this table. See *Table form field - Network data type* on page 657 for more information.

### Syntax

The syntax used to specify a single argument parameter. Note that the syntax is different for each *network data type* on page 657. Multiple different syntaxes can be supported for a particular network data type.

### Examples

Some examples according to the syntax specified for the network data type.

## 5.61. Form fields - Table - Network table

The contents of a network data table is collected from the network.

Examples: User accounts of an OU, domain or group.

The network data are collected automatically. The contents of the table changes dynamically when the network is updated. When the form is designed, the type of the network data and the *network call arguments* on page 655 are setup. At run-time, when the form is shown, the contents of the table is determined by accessing the network. The results are presented in the table. To prevent excessive network load, the table contents is stored for some *period of time* on page 651. In the screenshot below, a network table is shown with the **Common name** and **Username** of user accounts in a specific OU.



A network data type can have multiple columns, depending on the type of network data shown. Each table column can be linked to a variable. At run-time, when the user selects an item in the table and presses a submit button, the values of the selected item are stored in the corresponding values of the variables. The variables can then be used in a project script. In the screenshot shown above for instance, the column **Username** has been linked to the variable %UserName%. When the **Unlock account** submit button is pressed, the selected value **lmedeca** is stored in the variable %UserName%.

To start setting up or edit an existing form table, see *Table form field - Type* on page 660. The network data table is configured using several windows. The first one, network data type determines the type of the

network data. If this parameter is changed, the contents of other configuration settings is lost.



The following network data types are available:

Network data type	Description
User accounts of an organizational unit (OU)	Shows all user accounts of one or more OU's. Optionally, you can include the user accounts of child OU's of the configured OU's (Active Directory only).
User accounts of a global group	Shows all user accounts that are a member of one or more groups.
User accounts of a domain	Shows all user accounts that exist in a domain.
User accounts maintained on a computer	Shows all user accounts that are maintained on a computer, not necessarily a domain controller.

Note that the **Network data type** does not determine the data that must be collected from the network. It only determines the type of network data. Once the network data type is determined, you must specify the actual parameters or arguments that are used to collect the network data.

Example: the network data type **User accounts of an organizational unit (OU)** specifies that the type of network data equals user accounts that are collected from an OU. But the specification of the data type alone does not include the name of the OU from the user accounts must be collected.

The specification of the network data type determines the type of calls that will be executed by UMRA and the columns that can be shown in the corresponding network data table. But additional information is required to complete the network data table configuration. See *Table form field - Network call parameters* on page 655 for more information.

**See also:**

*UMRA tables* on page 9



## 5.62. Form fields - Table - Options

For a form table, there are some additional behaviour controls:

### **Table height specification - Specify the table height in numbers of rows shown**

A table in a form can contain any number of rows. The number of rows shown at a single point in time can be specified in this field. If the table contains more rows than the number specified, vertical scrolling is automatically enabled.

### **Multiple selection - Enable multiple selection**

Allows you to select multiple items in the table. When the end user selects multiple table entries in UMRA Forms, these selected entries will be stored in a multi- value variable.

### **Store indices of selected rows in table variable**

Allows you to store the indices of selected form table rows in the specified variable. The indices table is created for both single as multiple selection tables. The resulting table always has one column. The number of rows corresponds with the number of selected rows of the form table. For each row, the integer value is equal to the index of the selected form table row. The first form table row has an index of zero (0).

### **Double-click handling - When a table item is doubleclicked, select (click) the default (ENTER) button**

By default, this option is switched off. If it is activated, the default button **[ENTER]** is executed when the user doubleclicks a table entry.

### **User input state restore settings**

In this field, you can specify the items that define the user input state of the table. When a form is submitted by an end-user, the same form can be shown again. The contents of the form fields can be:

- **Reset to the initial value(s):** The field is shown as if the form is presented for the first time.
- **Restored from the previous form:** The field state (selection, entered text) is copied from the form that was submitted.

See *Form action - Return current form* on page 637 for more information.

### **Selection of table item(s)**

When a submit button is pressed and the same form is shown again, the

selection of the table is not changed if you select this option and if the user input state is restored for this table. See *Form action - Return current form* on page 637 on how to do this.

#### **Table scroll position**

When a submit button is pressed and the same form is shown again, the table scroll position is not changed if you select this option and if the user input state is restored for this table. See *Form action - Return current form* on page 637 on how to do this.

### **5.63. Form fields - Table - Row icon image**

For a form table, you can configure an icon to be shown in front of each row. The available icons are preconfigured and cannot be shown. If you want to present a different icon, please contact your UMRA reseller. In the window, the index of the current row icon image is selected. To change the selection, select another icon and click **Apply** or **OK**.

### **5.64. Form fields - Table - Type**

In a project form, a table form field can be included. The following table types are available:

<b>Table type</b>	<b>Subtype</b>	<b>Description</b>
Network table		Used to obtain the user accounts of an OU, global group, domain or single computer using an NT network call.
Fixed table		Used to display a list of fixed content in a table (e.g. a list of department names).
Generic table	LDAP	Used to show the results of an LDAP query in a table.
Generic table	File	Used to show the content of a text file (CSV file) in a table.

Generic table	Database	Used to show the results of a database query in a table.
Generic table	Variable	Used to show the content of a variable in a form table.

If you are not familiar with the use of tables in UMRA, then please read *UMRA tables* on page 9 first.

### 5.65. Form project - Form fields

The form fields make up a form. Different types of form fields are available to design a form. The form fields have several functions:

- explain the form to the user (static text, picture)
- let the user specify input data for the form project (input text, table, checkbox)
- initiate the execution of form actions - submit (button)
- make the form look appealing (picture, vertical space, status text)

To pass the form information to the script of a form project, variables are used. By specifying the value of a form field, the user sets the value of a variable. Some form fields can also be associated with an action. Actions include the execution of the script of a form project, setting a variable to a specific value etc.

The following table summarizes the usage of the different types of form fields. The column **Variable** indicates if a form fields of the corresponding type can be used to setup a variable. The column **Actions** shows if form actions can be executed by activating or configuring the corresponding field type.

Form field type	Variable	Actions	Description
<i>static text</i> on page 648	No	No	Used to describe the form and form fields.
<i>input text</i> on page 644	Yes	No	Used to specify a text. Examples: first, middle and last name, passwords, description fields, phone number etc. When the form is submitted, the text entered is stored as the value of the field variable. This variable can be used in the script of the form project.
<i>table</i> on page 660	Yes	No	Used to select an entry from a list. Examples of table contents: user accounts of an OU, domain or group, departments. Each column of the table can be linked to a variable. When the form is submitted, the value of the selected table entry is stored as the variable value. These variable can be used in the script of the form project.
<i>checkbox</i> on page 640	No	Yes	Used to enable or disable a specific function. Examples: disable an account, create and Exchange mailbox. The action <i>Form action - Set variable value</i> on page 638 can be used to pass to state of the checkbox to the form script.
<i>radio buttons</i> on page 647	Yes	Yes	Used to offer a set of predefined options.
button	No	Yes	Used to submit or reset the form. When a form is submitted, a number of actions can be executed. See <i>Form action - General</i> on page 635 for more information.

<i>picture on page 646</i>	No	No	Any picture can be embedded in the form to clarify the purpose of the form, make the form according to the company standards etc.
vertical space	No	No	Used to create some vertical spacing between form fields.

### 5.66. Function modules

All features and functions of User Management are divided into function modules and interface modules. The function modules are used to group related script actions. Each function module contains a number of script actions. Vice versa, each script actions belongs to one of the function modules. The following table shows all of the available script actions, and the corresponding function module for each of them.

Script action	Base function	Exchange function	Lotus Notes function	Advanced function
<b>User - Active Directory</b>				
Create user (AD)	V			
Create contact (AD)	V			
Get user (AD)	V			
Edit user (AD)	V			
Edit user logon	V			
Get user table (locked out/disabled)(AD)				V
Delete user (AD)	V			

Set user group memberships (AD)	V			
Remove user group memberships (AD)	V			
Move - rename user (AD)				V
Move cross domain (AD)				V
Create Exchange mailbox (2000/2003)		V		
Edit Exchange mailbox (2000/2003)		V		
Modify Exchange mailbox permissions (2000/2003)		V		
Move Exchange mailbox (2000/2003)		V		
Delete Exchange mailbox (2000/2003)		V		
Manage Exchange recipient mail addresses (2000/2003)		V		
<b>User - non Active Directory</b>				
Create user (no AD)	V			
Edit user (no AD)	V			

Edit user logon	V			
Delete user (no AD)	V			
Setup user global group memberships	V			
Add account to local group	V			
Remove group member	V			
Set primary group (non AD)	V			
<b>User - General user actions</b>				
Edit user logon	V			
Get user info	V			
Set Terminal Services user settings	V			
Get Terminal Services user settings	V			
Dial-in user settings	V			
<b>Active Directory</b>				
Create object (AD)				V
Delete object (AD)	V			
Get attribute (AD)	V			
Set attribute (AD)				V

Delete attribute value (AD)	V			
Set group memberships (AD)	V			
Remove specific group memberships (AD)	V			
Create group (AD)				V
Get object (AD)	V			
Search object (AD)				V
Get primary group (AD)	V			
Set primary group (AD)	V			
<b>File system</b>				
Create directory	V			
Get file/directory info	V			
Copy directory				V
Rename file or directory	V			
Setup security				V
Delete file(s)	V			
Delete directory	V			
Create share	V			
Edit share	V			
Delete share	V			



List files and/or directories	V			
<b>Other actions</b>				
Execute command line	V			
<b>Services</b>				
List services status	V			
Execute service command	V			
Configure service	V			
<b>Printer</b>				
List printer documents	V			
Execute print job command	V			
<b>LDAP</b>				
Setup LDAP session				V
Load LDAP modification data				V
Add directory service object (LDAP)				V
Modify directory service object (LDAP)				V
Rename directory service object (LDAP)				V

Delete directory service object (LDAP)				V
Search LDAP				V
<b>Lotus Notes</b>				
All Lotus Notes Actions in this folder			V	
<b>Variable actions (table)</b>				
Generate generic table				
Manage table data				V
<b>Variable actions (database)</b>				
Update database				V
<b>Variable actions (name generation)</b>				
Generate name(s)	V			
<b>Variable actions (Variable operations)</b>				
Set variable	V			
Set encrypted variable	V			
Split variable	V			
Format variable value	V			
Update numeric variable	V			

Convert value of variable	✓			
Convert text to date/time	✓			
Convert to multi-value variable	✓			
Manage multi-text value variable	✓			
Merge multi-text variable value	✓			
Export variables	✓			
Delete variable	✓			
Encrypt text	✓			
Generate random number	✓			
Generate password	✓			
Log variables	✓			
<b>Variable actions (Programming)</b>				
Map variable	✓			
Go to label	✓			
If-Then-Else				✓
Execute script				✓
For-Each				✓
Delay	✓			
No operation	✓			
<b>Variable actions (Mail)</b>				

Send mail message				V
-------------------	--	--	--	---

**See also:**

*License model* on page 707

*Interface modules* on page 675

## 5.67. Generic table - Introduction

The possibility of using generic tables is one of the most powerful features in UMRA. The use of generic tables has several key advantages:

**Easy selection of data**

First of all, it allows you to create very user friendly solutions for delegating user management tasks by offering table data in a form which the end user only needs to select. The table data could either be the result of a *database query* on page 672, an *LDAP query* on page 672 or a *variable* on page 672. A wide range of script actions can then be applied to the selected data.

**Link to other information systems**

The data you can retrieve in a generic table is not limited to Active Directory. With UMRA you can establish a link between Active Directory and any other database (SAP, SQL, Oracle, etc.) containing relevant user resource information. Using UMRA, record sets can also be updated. When an employee leaves the company for instance, UMRA could update all associated resources.

**Next step:**

*Choosing a table type* on page 672

**See also:**

*Form fields - Table - Fixed data* on page 654

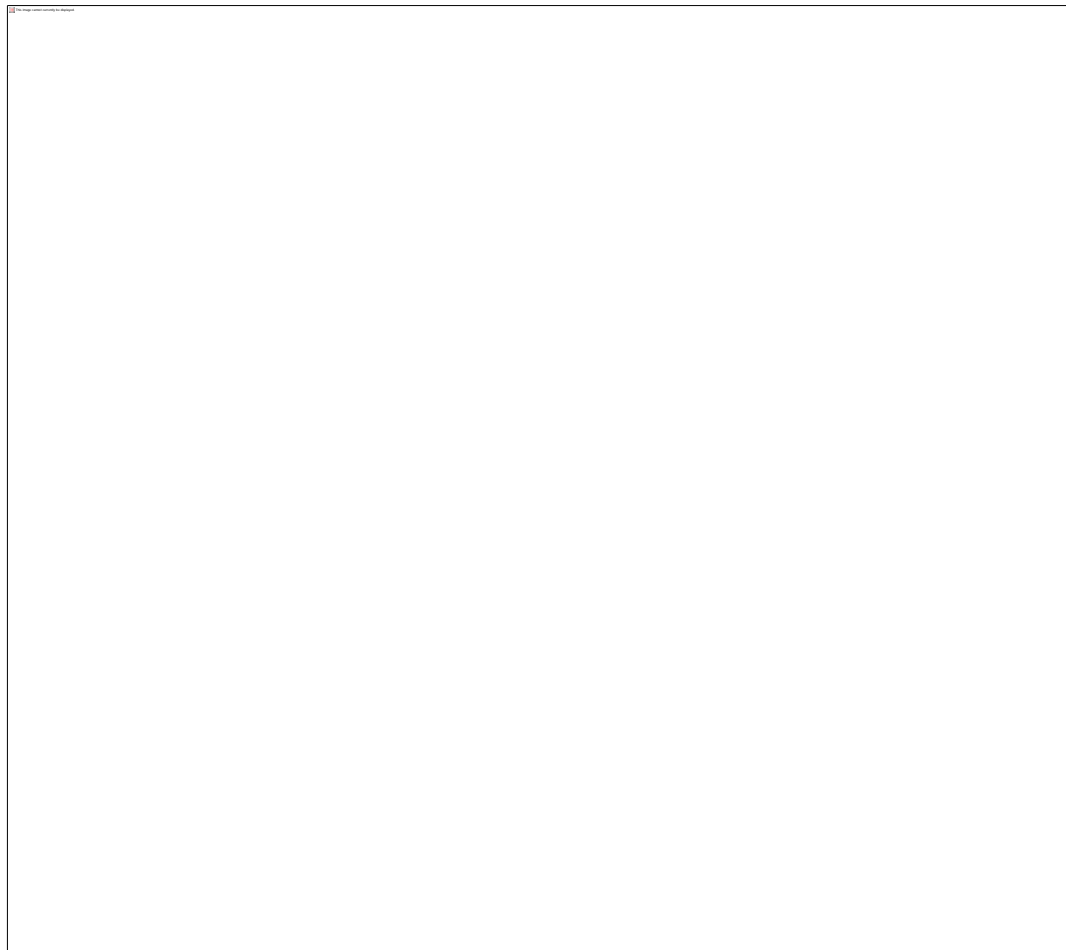
*Form fields - Table - Network table on page 657*

## 5.68. Generic table - Run test

In this window, you can run a test to ensure that the generic table data are correctly retrieved as a result of the database query, LDAP query or text file import.

### Running a test

Click the **Test** button to start the test. The generated table data will be displayed in the **Table data** section (see screenshot below)



**Run test on UMRA Service**

Use this option to check if the search request can also be handled correctly by the UMRA service. If the UMRA service does not have sufficient security privileges to run the query, you will receive an error.

For instance, if a database query is run against an MDB file which is located on a share to which the UMRA service has no access, you will receive the following error message:



## 5.69. Generic table - Table type

A generic table can hold data from various different sources:

- a file - Used to store the content of a CSV file in a table
- a database - Used to store the results of a database query in a table;
- an LDAP query - Used to store the results of an LDAP query in a table;
- variable - used to store the content of a variable in a table

These types are listed in the **Table type** dropdown list.

### File

The content of a text file (CSV) file can be converted to a generic table and shown in a form.

To setup a generic table containing data from a text file, you need to specify how the text file should be read.

See also *Specify file input data* on page 756.

### LDAP query

Using an LDAP query, you can specify which objects you would like to retrieve from the Active Directory (list of users, list of groups, last logon, etc.). These objects can be shown in a table. Selected table entries can be linked to a variable and used as input for an UMRA project script.

To setup a generic table containing an LDAP query, the following information has to be specified:

1. *LDAP binding* on page 694,
2. *LDAP filter* on page 697
3. *LDAP attributes* on page 689.

#### **Database query**

Many user related data are stored outside the Active Directory, possibly in another information system (e.g. a list of departments). Using a generic table, these data can be accessed and combined with information from Active Directory or other directory services supporting LDAP..

#### **Example**

Imagine a company X where the administrator would like to see the relation between user groups and departments. Based on this information he wants to perform certain actions such as removing group memberships, adding group memberships, etc. The user group data are stored in the Active Directory, but the relation between user groups and departments is stored in an MS Access database. Using a generic table in UMRA, these data can be accessed and queried.

To setup a generic table containing an LDAP query, the following information has to be specified:

1. *Database* on page 626
2. *Database query* on page 626

#### **Variable**

Many script actions collect data which are stored in table format in a variable (e.g. List printer documents and List services status). The content of these variables can be shown in a generic table, type Variable. If other table data manipulations are necessary for the required output, you can use the *Manage table data* on page 528 script action.

To setup a generic table of the Variable type, you need to specify the variable containing the table data and the columns for the table. See also *Specifying the table type Variable* on page 776

**See also:**

*Table form field - Generic table* on page 655

*UMRA tables* on page 9

*UMRA Basics* on page 3

## **5.70. Generic table - Column names**

Specify here the names for the columns for the resulting table. The first name listed will be used for the first column, the second one for the second column, and so on.

If no names are specified, default column names are used ("Column\_01", "Column\_02", etc) , unless the column names are already determined by other means. For instance, if a option "First line contains headers" is specified when creating a table from a file, the column names are read from the file instead.

The column names can be used in several script actions to refer to a particular column of the table.

## **5.71. Generic table - Variable**

The table content of a generic table can be stored in a variable. This variable can be used in the project script.

**See also:**

*UMRA tables* on page 9

*UMRA Basics* on page 3



## 5.72. Interface modules

An interface module specifies the User Management Resource Administrator interface that can be used to execute projects and scripts. At this moment, 4 interface modules exist:

1. **UMRA Mass module:** Mass projects to create-update-delete user accounts and resources in Active Directory and NT4-local computer networks in bulk. Mass projects are executed using the graphical UMRA Console application.
2. **UMRA Forms Module:** Form projects to execute any script to create-update-delete user accounts and resources in Active Directory and NT4-local computer networks. Form projects are executed by the UMRA Service. The forms are shown in the UMRA Forms application and managed using the UMRA Console application.
3. **UMRA Automation Module:** Supports the execution of UMRA Form and UMRA Mass projects through command-line interfaces for mass and by external applications through the COM object model. Also required for the execution of projects by means of the UMRA **Task scheduler**.
4. **SSRPM Module.** Allows the SSRPM (Self Service Reset Password Management) program of Tools4ever to communicate with the umra service.

### See also:

*License model* on page 705

*Function modules* on page 663

*License code* on page 705

## 5.73. LDAP attributes - Attribute specification

Apart from selecting a default attribute settings (see *LDAP search - Attributes* on page 689), you can also specify the required attribute yourself in this window.

1. Click the **Add** button. The following dialog box will appear:



1. Select the LDAP display name of the attribute in the **LDAP name** list box. This list includes the names of the most commonly used attributes of a user object:

LDAP name	Description
c	The country/region in the address of the user. The country/region is represented as the 2- character country code based on ISO-3166.
cn	The name that represents an object. Used to perform searches.
co	The 'co' (Friendly Country Name) attribute specifies names of countries in human-readable format. It is commonly used in conjunction with the 'c' (Country Name) [Schema] attribute (whose values are restricted to the two-letter codes defined in [ISO3166]).
company	The user's company name.
countryCode	Specifies the country code for the user's language of choice. This value is not used by Windows 2000.
department	Contains the name for the department in which the user works.
description	Contains the description to display for an object. This value is treated as single-valued by the system.
displayName	The display name for an object. This is usually the combination of the users first name, middle initial, and last name.
distinguishedName	Same as the Distinguished Name for an object. Used by Exchange.
facsimileTelephoneNumber	Contains telephone number of the user's business fax machine.
givenName	Contains the given name (first name) of the user.

homeDirectory	The home directory for the account. If homeDrive is set and specifies a drive letter, homeDirectory must be a UNC path. Otherwise, homeDirectory is a fully qualified local path including the drive letter (e.g. "c:\directory\folder"). This value can be a null string.
homeDrive	Specifies the drive letter to which to map the UNC path specified by homeDirectory. The drive letter must be specified in the form "<DriveLetter>:" where <DriveLetter> is the letter of the drive to map. The <DriveLetter> must be a single, uppercase letter and the colon (:) is required.
homePhone	The user's main home phone number.
info	The user's comments. This string can be a null string.
initials	Contains the initials for parts of the user's full name.
ipPhone	The TCP/IP address for the phone.
l	Represents the name of a locality, such as a town or city.
lastLogon	The last time the user logged on. This value is stored as a large integer that represents the number of 100 nanosecond intervals since January 1, 1601 (UTC). A value of zero means that the last logon time is unknown.
IDAPDisplayName	The name used by LDAP clients, such as the ADSI LDAP provider, to read and write the attribute using the LDAP protocol.
mail	The list of email addresses for a contact.
memberOf	The distinguished name of the groups to which this object belongs.
mobile	The primary cell phone number.
pager	The primary pager number.
physicalDeliveryOfficeName	Contains the office location in the user's place of business.
postalCode	The postal or zip code for mail delivery.

postOfficeBox	The P.O. Box number for this object.
profilePath	Specifies a path to the user's profile. This value can be a null string, a local absolute path, or a UNC path.
sAMAccountName	The logon name used to support clients and servers running older versions of the operating system, such as Windows NT 4.0, Windows 95, Windows 98, and LAN Manager. This attribute must be less than 20 characters to support older clients.
scriptPath	This attribute specifies the path for the user's logon script. The string can be null.
sn	This attribute contains the family or last name for a user.
st	The name of a user's state or province.
streetAddress	The street address.
telephoneNumber	The primary telephone number.
title	Contains the user's job title. This property is commonly used to indicate the formal job title, such as Senior Programmer, rather than occupational class, such as programmer. It is not typically used for suffix titles such as Esq. or DDS.
userAccountControl	Flags that control the behavior of the user account.
userPrincipalName	This attribute contains the UPN that is an Internet-style login name for a user based on the Internet standard RFC 822. The UPN is shorter than the distinguished name and easier to remember. By convention, this should map to the user e-mail name. The value set for this attribute is equal to the length of the user's ID and the domain name.

userWorkstations	Contains the NetBIOS or DNS names of the computers running Windows NT Workstation/Windows 2000 Professional from which the user can log on. Each NetBIOS name is separated by a comma. The NetBIOS name of a computer is the sAMAccountName property of a computer object. Multiple names should be separated by commas.
wWWHomePage	The primary web page.

You are not limited to using the above listed attributes. For a full list of attributes, see [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes\\_all.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes_all.asp)  
[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes\\_all.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes_all.asp) or check the MSDN library.

#### Attribute value conversions

Some attributes contain values which require data conversion. For more information about this topic, see *LDAP attributes - Data conversion* on page 679 and *LDAP attributes - Data conversion routine* on page 680.

### 5.74. LDAP attributes - Data conversion

Some attributes contain values which are hard to interpret. The attribute lastLogon for instance, which is used to check when a user last logged on, returns a value which represents the number of 100 nanosecond intervals since January 1, 1601 (UTC). A value of zero means that the last logon time is unknown. The following screenshot gives you an idea what this looks like:



In such cases you would probably want to present these values in a more user friendly way. This can easily be achieved using data conversions.

### 5.75.

#### Example - Specifying data conversion for the lastLogon attribute

1. Select the **Attribute** tab and click the **Add** button. This will bring up the **LDAP Attribute specification** dialog box

2. In the **LDAP name** field, enter "lastLogon" (this attribute is not included in the default list of attributes)
3. Click the **Setup...** button under **Data conversion**:



4. Click the **Add** button in the **Data conversion** dialog box



5. Select the operation **Convert large integer (100ns,1-1-1601) to date-time (last logon)**.



6. Click **OK**. The data conversion routine is now added to the Data conversion routines window.



7. Click **OK** twice to return to the **Setup Generic table** dialog box. Note that the lastLogon attribute has now been added and that data conversion has been set to "Yes".



8. Select the **Run test** tab and click the **Test...** button. You will see that the original value representing the last logon for the Administrator has been converted to an understandable date-time format.



For more information on data conversion routines, see *LDAP attributes - Data conversion routine* on page 680.

## 5.76. LDAP attributes - Data conversion routine

In UMRA, data conversion routines are used to present returned attribute values in a more user- friendly way. In *LDAP attributes - Data conversion* on page 679 an example was given on how to convert large

integers to a date-time format. In total UMRA offers three data conversion routines:

1. Perform logical AND on the input value and specified argument
2. *Convert large integer (100ns, 1-1-1601) to date-time (last logon) on page 679*
3. Convert large integer to specified text if zero.

In this topic we will describe the use of operation 1 and 3 and zoom in on some other options in the Data conversion routine dialog box.

#### **Perform logical AND on the input name and specified argument**

The routine Perform logical AND on the input value and specified argument is used to evaluate so called *bitmask attribute values* on page 697. A bitmask attribute is a single attribute that contains multiple properties and property values. For the sake of clarity, just consider a bitmask to be a bank of switches, with each switch representing a different property. If the switch for Account is disabled is on, then the account is disabled. If the switch is on, the user account is enabled. The only difficult part is that these "switches" do not have intuitive names such as "Account disabled". Instead, they have hexadecimal values like &H0040.

The userAccountControl attribute for instance, holds the following properties and hex values:

Property	Value
Logon script will be executed	&H0001
Account is disabled	&H0002
Account requires a home directory	&H0008
Account is locked out	&H0010
Account does not require a password	&H0020
User cannot change password	&H0040
Encrypted text password allowed	&H0080
Account password never expires	&H10000
Smartcard required for logon	&H40000

Password has expired	&H800000
----------------------	----------

If you want to create a generic table which returns a list of all users with a disabled account, you need to do the following:

1. Select the **Attribute** tab and click the **Add** button. This will bring up the **LDAP Attribute specification** dialog box.
2. In the **LDAP name** field, select the userAccountControl attribute and enter "Disabled account" as the display name.



3. Click the **Setup...** button under **Data conversion**.
4. Click the **Add** button in the **Data conversion** dialog box.
5. Select the operation **Perform logical AND on the input value and specified argument**.

This operation requires that you specify the value of the property as an argument in decimal format (not in hex format). The table above shows that the hex value for the property "Account is disabled" is "&H0002". This hexadecimal value needs to be converted to a decimal before it can be entered as an argument. This conversion can be done in any Windows calculator. In this case, the decimal value is "2".

6. Enter 2 in the **Argument** text field.



7. Click **OK**. The data conversion routine is now added to the **Data conversion routines** window.



8. Click **OK** twice to return to the **Setup Generic table** dialog box. Note that the lastLogon attribute has now been added and that data conversion has been set to "Yes".
9. Select the **Run test** tab and click the **Test...** button. Users with a disabled account are now displayed in the **Disabled accounts** column ("Yes" is disabled, "No" is enabled).





### Converting a large integer to specified text if zero

The lastLogon attribute contains a value which tells us when a user last logged in. If the query returns a zero, it means that the last logon time is unknown. By default, this is not displayed. In the following screenshot for instance, the last logon time for the users "Guest" and "Frédéric Vallenet" is unknown, but not displayed as such.



By making use of the routine **Convert large integer to specified text if zero**, we can include a string to be displayed when the value of the lastLogon attribute is zero.

1. Select the **Attribute** tab and click the **Add** button. This will bring up the **LDAP Attribute specification** dialog box.
2. In the **LDAP name** field, enter "lastLogon" (this attribute is not included in the default list of attributes).
3. Click the **Setup...** button under **Data conversion**.
4. Click the **Add** button in the **Data conversion** dialog box.
5. Select the operation **Convert large integer to specified text if zero**.
6. In the **Argument** field, enter "Unknown". This will display the string "Unknown" for all user objects where the lastLogon attribute value is zero,. Click **OK**.
7. Click **OK**. The data conversion routine is now added to the **Data conversion routines** window.



8. Click **OK** twice to return to the **Setup Generic table** dialog box. Note that the lastLogon attribute has now been added and that data conversion has been set to "Yes".
9. Select the **Run test** tab and click the **Test...** button. You will see that the zero value for the two user objects is now displayed as "Unknown".



### 5.77. LDAP Directory Service - Encrypt input

Important: This script action is one of a series of script actions to manage LDAP directory services using UMRA. If you are not familiar with this topic yet, then please read *Managing LDAP directory services using UMRA* on page 25 first.

In this window you can specify the password of the user with property "User name".

**Normal value:** Here you can enter a user password directly. When you click OK, the password will be automatically encrypted.

**Encrypted value:** If the password has been encrypted using the Set encrypted variable script action, this variable can be entered or selected in the Encrypted value field.

In an UMRA script, the passwords are always stored encrypted. When an LDAP session is established however, the password is automatically decrypted and there will be two possible options:

1. **Non-secure communication (No SSL encryption)** - All communication with the LDAP Server and the UMRA software is not encrypted. Authentication is accomplished using an account name and the password which was encrypted in UMRA will be decrypted automatically and sent as clear text. Although simple to implement, this option is not recommended because of security reasons. The option can be used for testing purposes.
2. **Secure with SSL (SSL Encryption flag set to "Yes"** - All communication between the LDAP client, e.g. the UMRA software and the LDAP Server is encrypted using the SSL standard. This option is recommended and secure. All data is sent encrypted.

#### Generating your own key for encryption and decryption

UMRA uses the same key for encryption and decryption which is automatically generated when you install the UMRA service. For security reasons, you may decide to generate your own key, in which case you must ensure that the key which is generated on the UMRA Console side is identical to the one on the UMRA service side. Ho to do this, is described in the following procedure.

*Generating a key on the UMRA Service side*

1. In Windows, select **Start-->All Programs-->Administrative Tools-->Active Directory Users and Computers**. Select the **\Users** folder.
2. Right-click the **UMRA Service** (called UmraSvcAccount) and select the **Reset Password** command.



3. Enter a new password in the **Reset Password** dialog box and confirm your password. Note that you can only do this if you are currently logged in with an administrator password.



4. Log off by selecting **Start-->Log Off**

Next, we need to log on using the UMRA Service account and the password we have just entered to create a new registry key.

5. Press **Ctrl-Alt-Delete** and enter the **UMRA Service** account (UMRASvcAccount) and password:
6. Start the registry editor by selecting **Start-->Run** and entering "**Regedit**".



7. Create the following key:  
**HKEY\_CURRENT\_USER\Software\tools4ever\UMRA\Communication**
8. In the **Communication** folder, create a new String value called "Key" and enter a password. This key and the password should be exactly the same on the **UMRA Console** side!!!



9. Log off.

*Generating a key on the UMRA Console side*

1. Log in using an account with administrative rights. Repeat steps 6-8 as described under **Generating a key on the UMRA Service side**. The key has now been successfully changed.

**See also:**

*Script Action: Set encrypted variable* on page 546

## 5.78. LDAP Directory Service - LDAP Search

Important: This script action is one of a series of script actions to manage LDAP directory services using UMRA. If you are not familiar with this topic yet, then please read *Managing LDAP directory services using UMRA* on page 25 first.

The LDAP Search window is used to specify the LDAP search.

**Session**

The variable representing the LDAP Session that is initialized with action Setup LDAP session.

**Result**

The name of the variable that is used to store the result of the search. The search result is always stored as a table. The variable does not need to exist when the action is executed. If it does not exist, the old value is overwritten.

**Base (DN)**

The distinguished name of the directory service tree where the search should start. The search is executed at the specified base, and optionally in the immediate or all subtrees of the directory service.

**Filter**

The specification of the filter to perform the search. The standard search specification according to RFC2254 can be used to execute the search.

**Scope**

Base only	Limits the search to the specified base only. The maximum number of matching directory service items is 1.
One level	The search is performed in all entries of the first level below the base entry, excluding the base entry.
Subtree	The search is performed in the base entry and all levels below the base entry.

Time out interval	When enabled, the specified value is the time-out value of the LDAP search and the operation time. If disabled, no time-out value is used.
Size limit	When enabled, the maximum number of matching values is limited to the specified value. When disabled, the maximum number of items is not limited.

*Managing LDAP directory services using UMRA on page 25*

Important: This script action is one of a series of script actions to manage LDAP directory services using UMRA. If you are not familiar with this topic yet, then please read *Managing LDAP directory services using UMRA* on page 25 first.

The result of the search is a table. In the table, the rows correspond with matching directory service items. Each column corresponds with an attribute. The distinguished name is by default stored in the last column of the table. So the example shown in the figure above will result in a table with 4 columns. The distinguished name is normally used to identify a directory service item.

Note: The column names are not stored as part of the table data. If the variable is in a form to show the table data, the column names need to be specified as part of the table form field specification. See *Variable generic table* on page 776 for more information.

**See also:**

*Variable generic table* on page 776

*Script action: Setup LDAP session* on page 5

*Script action: Load LDAP modification data* on page 7

*Script action: Add directory service object (LDAP)* on page 10

*Script action: Modify directory service object (LDAP)* on page 10

*Script action: Delete directory service object (LDAP)* on page 11

*Script Action: Search LDAP*

[http://www.tools4ever.com/resources/manual/usermanagement6/script\\_action\\_search\\_ldap.htm](http://www.tools4ever.com/resources/manual/usermanagement6/script_action_search_ldap.htm) \t t4ehelppopup

## 5.80. LDAP Directory Service - Setup LDAP modification data

Important: This script action is one of a series of script actions to manage LDAP directory services using UMRA. If you are not familiar with this topic yet, then please read *Managing LDAP directory services using UMRA* on page 25 first.

The **Setup LDAP modification data** window is used to specify the values of an attribute.

For the attribute you need to specify the following:

1. **Type of modification:** Either Add, Delete or Replace, depending on the required type of modification.
2. **Add:** add the specified values to the attribute. Existing attributes values are not removed. If the attribute already contains one of the specified values, an error occurs.
3. **Delete:** delete the specified value from the attribute. If the specified value is not a value of the attribute, an error occurs.

4. **Replace:** delete all of the existing attribute values and add the specified values to the attribute.
5. **Type of data:** The type of the data, either text or binary. Almost all attribute values, including text, numbers, Boolean flags, date and time values can be specified using text.
6. **Data specification:** The attribute values. These values can be specified as fixed values or variables.

**See also:**

*Managing LDAP directory services using UMRA on page 25*

*Script action: Setup LDAP session on page 5*

*Script action: Load LDAP modification data on page 7*

*Script action: Add directory service object (LDAP) on page 10*

*Script action: Modify directory service object (LDAP) on page 10*

*Script action: Delete directory service object (LDAP) on page 11*

*Script Action: Search LDAP*

[http://www.tools4ever.com/resources/manual/usermanagement6/script\\_action\\_search\\_ldap.htm](http://www.tools4ever.com/resources/manual/usermanagement6/script_action_search_ldap.htm) \t t4ehelppopup

## 5.81. LDAP search - Attributes

**Previous actions:**

1. *Specifying the table type on page 655*
2. *Specifying the LDAP binding method on page 694*
3. *Defining the LDAP filter on page 697*

**General**

Each object in Active Directory has a set of attributes, defined by and depending on its type and class. Using the LDAP filter you have filtered

on some objects representing single entities (users, computers, printers, applications, etc.) and their attributes. In the Attributes tab you need to define which attributes you wish to return for the filtered objects.

### Specifying LDAP attributes in UMRA

In the LDAP filter you have defined which objects you want to retrieve. The next step is to define the attributes you want to have returned for these objects. In UMRA, you can either specify an LDAP attribute yourself or select a default attribute setting.

#### 1. Selecting a default attribute setting



You can select one or more predefined attributes from the **Default attribute settings** list to include in your query. The table below shows the corresponding LDAP name which is inserted in the **Attributes** window when you click the **Set** button.

If you select the Attribute example setting	The following attributes are returned	Display Name	Description
A. Users - general information	cn	Name	Name that represents an object
	description	Description	Contains the description to display for an object.
B. Users - locked out, disabled	A plus the following:		
	userAccountControl	Locked out	Flags controlling the user account behaviour.
	userAccountControl	Disabled	



C. Users - locked out, disabled + more options	A+B+D		
D. Users - password options	A plus the following:		
	userAccountControl	User must change password at next logon	
	userAccountControl	User cannot change password	
	userAccountControl	Password never expires	
E. Users - full details	C plus the following		
	profilePath	Profile path	Specifies a path to the user's profile. This value can be a null string, a local absolute path, or a UNC path.
	scriptPath	Script path	The path to the user's logon script
	home drive	Home directory drive	Specifies the drive letter to which to map the UNC path specified by homeDirectory.

	homeDirectory	Home directory	The home directory for the account. If homeDrive is set and specifies a drive letter, homeDirectory must be a UNC path.
F. Users - names	cn	Name	
	displayName	Display name	The display name for an object. This is usually the combination of the users first name, middle initial, and last name.
	givenName	First name	First name of the user
	initials	Initials	Contains the initials for parts of the user's full name.
	sn	Last name	Contains the last name for a user
	sAMAccountName	SAM Account Name	The logon name used to support clients and servers running older versions of the operating system, such as Windows NT 4.0, Windows 95, Windows 98, and LAN Manager.

	userPrincipalName	User Principal Name	This attribute contains the UPN which is an Internet- style login name for a user based on the Internet standard RFC 822. By convention, this should map to the user e-mail name. The value set for this attribute is equal to the length of the users ID and the domain name.
	distinguishedName	Object Distinguished Name	Same as the Distinguished Name for an object. Used by Exchange.
G. Users - last logon	A plus the following:		
	lastLogon	Last Logon	The last time the user logged on. This value is stored as a large integer that represents the number of 100 nanosecond intervals since January 1, 1601 (UTC). A value of zero means that the last logon time is unknown.

For a complete overview of Active Directory attributes, please see the Microsoft website

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes\\_all.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes_all.asp)

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes\\_all.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adschema/adschema/attributes_all.asp)

## **2. Specifying an LDAP attribute**

Apart from selecting a default attribute setting based on its description (Users - last logon, Users - names) you can also specify an attribute yourself. For more information, see *LDAP attributes - Attribute specification* on page 675.

### **Next action:**

*Generic table - Run test* on page 671

### **See also:**

*UMRA tables* on page 9

## **5.82. LDAP search - LDAP binding**

### **Previous actions:**

1. *Specifying the table type* on page 655

Starting with Windows 2000, the LDAP provider is used to access Active Directory. This binding method requires a binding string, which can be defined in three different ways in UMRA:

### **1. Global Catalog - Binding to the global catalog**

The global catalog is a searchable master index containing directory data of all domains in a forest. It contains an entry for every object in the forest, but it does not include all properties of each object. The Global Catalog is used to improve the response time of LDAP searches. The properties included in the Global Catalog are generally useful for searches and are considered static (dynamic properties would cause excess replication). Note that if you are searching for a property which is

not included in the Global Catalog, the search will only be conducted in the **current** domain.

In UMRA, there are two ways of binding to the Global Catalog. You can either select the option **Global Catalog** or select the **Binding String** option and enter the binding string manually.

## 2. Active Directory root - Binding to the Active Directory root

This option will bind to the Active Directory root of a domain controller. The Active Directory contains all the network information for the forest.

### 3a. Binding string - Binding to Active Directory using a binding string

You can also enter a binding string manually. This binding string is the AdsPath of an object in Active Directory, consisting of the LDAP provider moniker (LDAP://) appended to the Distinguished Name of the object. The Distinguished Name specifies both the name and the location of an object in the Active Directory hierarchy.

The Distinguished Name consists of a series of components separated by commas. Each component consists of a moniker, an equals sign, and the name of the component. Below are examples of statements that bind to objects with the LDAP provider. The binding string is the string in quotes.

- "LDAP://cn=John Smith,ou=Sales,dc=MyDomain,dc=com"
- "LDAP://cn=Test2,cn=Users,dc=MyDomain,dc=com"
- "LDAP://cn=Engr,ou=East,dc=MyDomain,dc=net"
- "LDAP://ou=Sales,ou=East,dc=MyDomain,dc=MyFirm,dc=com"

In the table below, the various components are explained:

Component	Description
LDAP	The provider (case sensitive)
cn=John Smith,ou=Sales,ou=East,dc=MyDomain,dc=net	Distinguished Name of user "John Smith"
cn=John Smith	Relative Distinguished Name of user "John Smith"
dc=MyDomain,dc=com	DNS domain name (MyDomain.com)

cn=Users	Relative Distinguished Name of container "Users"
ou=Sales	Organizational Unit where user "John Smith" resides
cn	Common Name
ou	Organizational Unit
dc	Domain

As an example, the Distinguished Name "cn=John Smith,ou=Sales,dc=MyDomain,dc=com" has four components. The first (lowest level) component of the Distinguished Name is the Relative Distinguished Name (RDN) of the object. In this case, the RDN is "cn=John Smith". The RDN of an object is the name of the object in its container. The remainder of the components are the Distinguished Name of the container, which is the parent of the object. In this case, the object "cn=John Smith" is in the container whose Distinguished Name is "ou=Sales,dc=MyDomain,dc=com". In this case, the parent container is an organizational unit. The parent of the "ou=Sales" organizational unit is the domain "MyDomain.com". This domain has domain components "dc=MyDomain" and "dc=com". The full DNS name of the domain is "dc=MyDomain,dc=com".

Container objects can be containers, organizational units, or domains. Container objects are objects that can "contain" other objects, such as user objects, group objects, and computer objects. Group objects are not containers. Groups can have members, but the members are not children of the group object.

### 3b. Binding string - Binding to the Global Catalog using a binding string

You can also bind to the server holding the global catalog using the GC provider. "GC:" uses the LDAP provider to bind to the Global Catalog service to execute fast queries.

The syntax is:

**GC://<host name>/<object name>**

where <host name> specifies the (DNS) name of the server holding the Global Catalog and <object name> represents a specific Active Directory object.

**Next actions:**

1. *Defining the LDAP filter* on page 697
2. *Defining LDAP attributes* on page 689
3. *Setting LDAP options (optional)* on page 702
4. *Specifying a table variable (optional)* on page 674
5. *Running a search filter test (optional)* on page 671

**See also:**

*UMRA tables* on page 9

## 5.83. LDAP search - LDAP Filter

**Previous actions:**

1. *Specifying the table type* on page 655
2. *Specifying the LDAP binding method* on page 694

**LDAP filter - General**

Once you have specified the data source for your generic table (see *Table form field - Generic table* on page 655 ) and the LDAP binding method, (see *LDAP search - LDAP binding* on page 694) you will need to specify which objects you would like to retrieve by defining a search filter. A search filter can be defined as a clause specifying the conditions that must be met for records to be included in the resulting record set.

### LDAP filter - Syntax

As mentioned above, you define all conditions that must be met for an object in the search filter. A condition takes the form of a conditional statement, such as "(cn=TestUser)". Each condition must be enclosed in parenthesis. In general, a condition includes an attribute and a value, separated by an operator.

Conditions can be combined using the following operators (note that the operators "<" and ">" are not supported).

Operator	Description
=	Equal to
~=	Approximately equal to
<=	Less than or equal to
>=	Greater than or equal to
&	AND
	OR
!	NOT

Conditions can also be nested using parenthesis. Furthermore, you can use the "\*" wildcard character in the search filter.

### The LDAP filter in UMRA

For the LDAP filter in UMRA you can either make a choice from a list of predefined search filters under **Example LDAP search filters** or enter your own search filter directly in the **LDAP Search filter** window.





To select all users for example, simply select the **All users** option and click the **Insert** button. The actual LDAP search syntax for this filter, "(objectClass=user)" will now appear in the **LDAP search filter** window.



#### Some examples of filtering actions

To	Use the following LDAP filter
Return all user objects except those whose surname attribute equals "Macintosh"	(&(objectClass=user)!(sn=Macintosh))
Return all user objects with a surname that starts with sm	(sn=sm*)
Return all contacts with a surname equal to Smith or Johnson	(&(objectClass=contact)( (sn=Bridges) (sn=Macintosh)))

Return all user objects with cn (Common Name) beginning with the string "Joe"	(&(objectCategory=person)(objectClass=user)(cn=Joe*))
Return all computer objects with no entry for description	(&(objectCategory=computer)(!description=*))
Return all user and contact objects	(objectCategory=person)
Return all group objects with an entry for description	(&(objCategory=group)(description=*))
Return all groups with cn starting with "Helpdesk" or "Admin"	(&(objectCategory=group)( (cn=Test*)(cn=Admin*)))
Return all users with "Password Never Expires" set	(&(objectCategory=person)(objectClass=user) (userAccountControl:1.2.840.113556.1.4.803:=65536)) The attribute userAccountControl is a bitmask attribute. See the section Bitmask attributes below for a detailed explanation.
Return all users with disabled accounts	(&(objectCategory=person)(objectClass=user) (userAccountControl:1.2.840.113556.1.4.803:=2)) The attribute userAccountControl is a bitmask attribute. See the section Bitmask attributes below for a detailed explanation.

Return all users with "Allow access" checked on the "Dial-in" tab of the user properties dialog of Active Directory Users & Computers. These are all users allowed to dial in. Note that "TRUE" is case sensitive (for this query to work, you need to bind to the Active Directory root)	(&(objectCategory=person)(objectClass=user)&(msNPAllowDialin=TRUE))
Return all user objects created after a specified date (01/01/2005)	(&(objectCategory=person)(objectClass=user)(whenCreated>=20050101000000.0Z))

Return all users that must change their password the next time they logon (for this query to work, you need to bind to the Active Directory root)	(&(objectCategory=person)(objectClass=user)(pwdLastSet=0))
---	--

**Next action:**

*LDAP search - Attributes* on page 689

**See also:**

*UMRA tables* on page 9

## 5.84. LDAP search - Options

**Previous actions:**

1. *Specifying the table type* on page 655
2. *Specifying the LDAP binding method* on page 694
3. *Creating an LDAP filter* on page 697

### LDAP search options

In this section you can define the scope of your LDAP search and some additional options for your LDAP search. The following sections describe in detail the various possible configurations.

Time limit options (by default not set)	Description
Maximum search time	Specifies the maximum time for the LDAP search. If the time limit is reached, the search is ended.
Page time limit	Specifies the amount of time the UMRA client waits for a result set before terminating the search request.
Size limit options (by default not set)	
Total size limit	Specifies the size of the result set. If the result set reaches the specified size, the result set is considered complete.
Page size limit	The maximum number of records to be processed by the domain controller and returned to the UMRA client before continuing the search.
Cache results options (set by default)	
Cache result	Specifies whether the result set should be cached to the client. For very large result sets, disabling caching will reduce memory consumption on the client.
Scope options	
Search subtree, including all the children and the base object (default)	The search includes the entire Active Directory structure below the search base

Search one level of the immediate children, excluding the base object	The search includes any immediate children (sub containers or OUs)
Search base object only (result contains one object maximum)	This means that only the search base object is included in the search and no child containers or OUs. The maximum number of objects returned is one..
Referral chasing options	
Never	
Subordinate referrals only	This option needs to be selected if the LDAP search requires proceeding into parts of the directory tree that are not stored on the current domain controller.
External referrals only	The LDAP search needs to follow up references to an LDAP directory on another domain.
Always	

### Referrals

Every domain controller holds information about the other domains in the forest in the domain controller's Configuration container. When an LDAP search in Active Directory requires action on objects that are located on another domain controller, the client is referred to a domain controller that holds the requested object. This way, clients can query the root domain and reach the appropriate domain controller without having to know the name or location of the child domain.

### See also:

*UMRA tables* on page 9

## 5.85. License code

A UMRA license code contains the following information:

1. The licensing model: *demo*, *domain/OU* or *site* on page 707;
2. Name of the domain/OU or site;
3. Maximum number of users (domain/OU license only)
4. *Function* on page 663 modules
5. *Interface* on page 675 modules

Dependent on the UMRA installation, UMRA license codes must be installed for one or more UMRA components:

Feature	UMRA applications used	UMRA License Code installed for
Run mass projects	UMRA console	UMRA console
Run delegated forms	<ol style="list-style-type: none"><li>1. UMRA console to setup form projects and manage the UMRA service</li><li>2. UMRA forms used by helpdesk employees to view and submit forms</li><li>3. UMRA service to execute submitted forms</li></ol>	UMRA console and UMRA service

Note: A single license code can contain any combination of function and interface modules.

### Required information to obtain a license

To obtain a license code for User Management Resource Administrator, you need to contact your UMRA reseller. Please note that you are required to pass some network specific information in order to obtain a valid license code. In most cases, you will need a license code for either an entire domain or an organizational unit. Note that such a code is also valid for all child organizational units in the domain - organizational unit. If you want to manage user accounts in multiple domains or organizational units that do not have a parent-child relationship, you will also need multiple license codes.

The following information is required in order to generate a license code:

1. The **name of the domain or organizational unit**. If you need a license for a domain (and all child organizational units), you can specify the domain name either in NETBIOS format (example: TOOLS4EVER) or use the DNS name (example: tools4ever.com). In order to license a organizational unit (and all child organizational units) you need to specify the name of the organizational units in the following form: [domain DNS name]/[name of organizational unit]. Examples:  
tools4ever.com/Development,  
tools4ever.com/Development/UserManagementTeam.
2. The **maximum number of user accounts of the domain or organizational unit**. You need to include all user accounts of child organizational units as well. The number should be specified as one of the available tier levels: 100, 200, 250, 500, 750, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000 (10k), 20k, 30k, 40k, 50k, 60k, 70k, 80k, 90k, 100k, 110k, 120k, 130k, 140k, 150k, 200k, 250k, unlimited. Select the nearest tier level that exceeds the expected maximum number of user accounts (examples: 180 -> 200, 4560 -> 5000, 37000 -> 40000).
3. The *function* on page 663 and *interface* on page 675 modules that must be supported by the license code.

#### Specification of a license code

A license code looks like this:

```
BEGIN_CODE
Ni4m6jZkCD-4kDG33rASG-SWF15Ym7em
SWFsY5dMm-x85WWny7ny-Efdm3D3mQF
745kFn77ny-B6EZQj8kyD-pXD2fDXmQ
Tools4ever
END_CODE
```

The code contains 6 lines of text, begins with BEGIN\_CODE and ends with END\_CODE. Lines 2,3 and 4 contain the actual license code with a total of 90 characters, split over 3 lines with 30 characters per line ( [10 chars] - [10 chars] - [10 chars] ). Line 5 contains the registered license



name. This can be the domain name, name of the organizational unit or the name of the site. The syntax of the license code is:

```
BEGIN_CODE  
[10-chars] - [10 chars] - [10 chars]  
[10-chars] - [10 chars] - [10 chars]  
[10-chars] - [10 chars] - [10 chars]  
[registered name]  
END_CODE
```

To configure a license code for UMRA console, start the application and selection menu option Help, License. The presented window shows all installed licenses. Press Add. Although not required, it is most convenient to copy-paste a license code into the section License code. If the license code is already in the clipboard (press Ctrl-C when the license code is selected in a text editor as notepad) you only need to press Paste. Once the license code is specified, press OK. The license code is now installed. Once the code is installed, press OK. When you no longer need a license code, it is advised to remove the license code.

For more information on how to setup a license code for UMRA service, see *UMRA service - license* on page 773.

**See also:**

*License model* on page 707

*Interface modules* on page 675

*Function modules* on page 663

## 5.86. License model

### Introduction

To setup a licensed version of User Management Resource Administrator you need to have one or more valid license codes. You can

obtain a license code for UMRA from your reseller. Note that the license codes of UserManagement 5.x (UserManagement Professional, Delegation and Import) cannot be used for User Management Resource Administrator.

#### Licensing model

User Management Resource Administrator supports three licensing models:

1. **Demo license** - A demo license has a limited lifetime. The default demo period is 30 days. During the demo period, script execution is limited to 5 times per session, which can be reset by restarting the application. The demo license supports all function modules for the Console Interface Module (see further). When started for the first time, a demo license is automatically installed.
2. **Domain** - Organizational Unit license: This is default license model when purchased. The license is related to a domain or organizational unit (and all child organizational units). This type of license grants access to the functions of User Management Resource Administrator as long as the managed user accounts are a member of the licensed domain or organizational unit. A Domain - Organizational Unit license is further based on the maximum number of user accounts that exist in the domain (organizational unit) and all child organizational units. If the actual number of users exceeds the maximum number of the license, the license is no longer valid and User Management Resource Administrator will not execute any script or project. (For more complex scenario's, additional licensing options are available. Contact your reseller for more information).
3. **Site license** - A site license grants access to the functions of User Management Resource Administrator, regardless of the number of user accounts managed with User Management Resource Administrator.

Once a license is installed, you can always change to another licensing model if you have a valid license code. All configuration settings are preserved in this procedure.

### Function and Interface modules

All features and functions of User Management Resource Administrator are divided into modules. Each module has its own specific feature set. Two types of modules are defined for UMRA:

1. **Function modules** - A function module contains a number of script actions. The script actions can be regarded as the instruction set of UMRA. Each script action belongs to a specific function module. For an overview of script actions and functions modules, see *Function modules* for more information. For more general information on script actions, see *UMRA Basics* on page 3.
2. **Interface modules** - An interface module contains one or more UMRA applications that are used to run the UMRA scripts. The currently available interface modules are: (1) UMRA Mass Module (2) UMRA Forms Module and (3) UMRA Automation Module.

### License code

An UMRA license is installed by the configuration of a license code. A license code contains information regarding the licensing model and the function and interface modules. When the demo version is expired, you need to install a license code in order to continue working with UMRA. A license code can be obtained from your UMRA reseller. For more information, see *License code* on page 705.

### See also:

*Interface modules* on page 675

*Function modules* on page 663

*License code* on page 705

## 5.87. License matrix

UMRA consists of the following software applications:

1. **UMRA Console** - The main application that is primarily used to manage all UMRA projects and manage the UMRA service. To use UMRA, you always start with the UMRA Console application.

2. **UMRA Service** - The UMRA service is used to execute delegated tasks. The UMRA Service is accessed through the UMRA Console, UMRA Forms and UMRA Automation software. See *UMRA Basics* on page 3 and *Getting Started* on page 3 for more information.
3. **UMRA Forms** - The Windows interface to show and submit delegated forms. The UMRA Forms application is most often used by helpdesk employees. The UMRA Forms application interfaces with the UMRA Service application directly. See *UMRA Basics* on page 3 for more information.
4. **UMRA Automation** - UMRA can be integrated with other employee management systems to automate Active Directory user account management tasks. For instance: When an employee leaves an organization and is excluded from an employee information system, Active Directory needs to be updated, by disabling or removal of the associated user account and network resources. With UMRA, the UMRA service can execute these tasks automatically when the employee information system is updated. See *UMRA Basics* on page 3 and *Integrate UMRA with other applications using COM* on page 75 for more information.

The combination of supported *functions* on page 663 and required *interface modules* on page 675 of User Management Resource Administrator is shown in the license matrix table below.

Function	UMRA software	Required Interface Modules license	Function Module license
Run mass projects with the graphical UMRA Console user interface	UMRA Console	UMRA Mass Module	Base Function Module Exchange Function Module Advanced Function Module

Run mass projects with the command line options of the UMRA Console application.	UMRA Console UMRA Automation	UMRA Automation Module	Base Function Module Exchange Function Module Advanced Function Module
Run form projects with UMRA Forms	UMRA Console UMRA Service UMRA Forms	UMRA Forms Module	Base Function Module Exchange Function Module Advanced Function Module
Run form projects from a command line	UMRA Console UMRA Service UMRA Automation	UMRA Automation Module	Base Function Module Exchange Function Module Advanced Function Module
Run form projects using COM objects (ASP, Office, etc.)	UMRA Console UMRA Service UMRA Automation	UMRA Automation Module	Base Function Module Exchange Function Module Advanced Function Module

**See also:**

*Function modules* on page 663

*Interface modules* on page 675

*License code* on page 705

## 5.88. Log information

User Management Resource Administrator logs extensive project information to the log window and log file. Each time a project is run, a

new log file is generated. In the log window, at the bottom of the screen, the same log information is written. To setup the various log options, select **Tools-->Options-->Log settings** from the main menu.



This window contains the following fields:

**Automatically show the log window when a job is started**

With the menu option **View-->Log Bar**, you can toggle the log bar on and off. When this option is selected, the log bar is automatically shown when a job is started.

**Reset the log window when a job is started**

Select this option if you want to clear the content of the log window each time a new job is started.

**Store log information in files - Log file directory**

When selected, the information is logged to files. For each User Management Resource Administrator session, a new log file is generated in the specified log directory. The name of the file has the following syntax:

**UMLOG\_mm\_dd\_yyyy.txt**

where mm,dd and yyyy represent the current month, day and year respectively.

**See also:**

*UMRA Basics* on page 3

*Getting Started* on page 3

Help on help

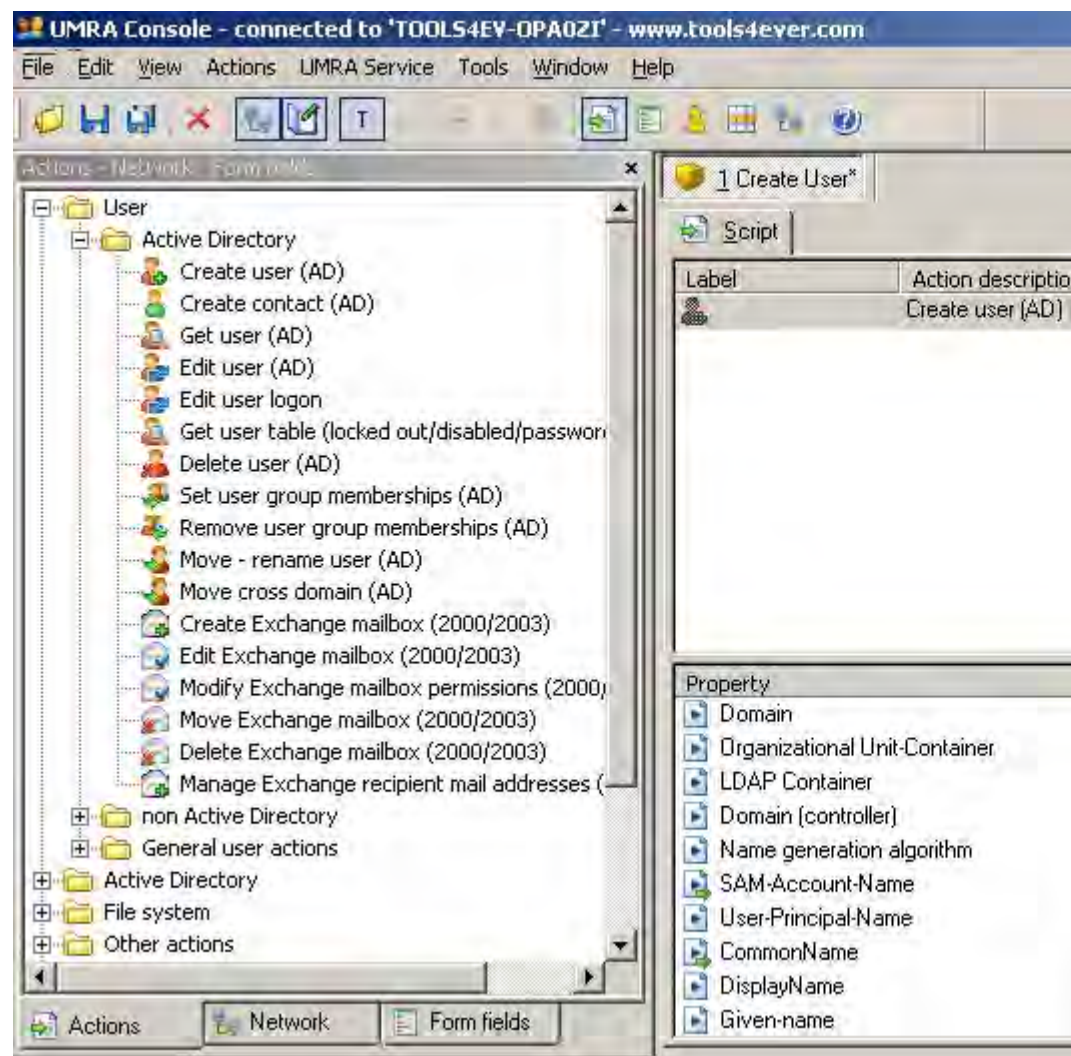
## **5.89. Manage script actions**

Every UMRA project contains at least a project script to perform a specific task. A project script consists of one or more script actions.

The available actions that can be added to a script are predefined and shown in the **Actions-Network-Form fields** window.

### Adding a script action to a project script

1. Click the **Actions** tab in the **Actions-Network-Form fields** window.
2. Select the required script action and drop it in the **Script** window of the project.



### Moving a script action within a project script

The position of a script action in a script is important. If the order is not correct, the script cannot be executed correctly. Example: if you create a user account and Exchange mailbox, the action to create the user must precede the action that creates the Exchange mailbox. To move a script

action in the script to another position, drag and drop the script action to the desired position.

#### **Deleting a script action from a project script**

To delete a script action from a script, select the action and press the Del key.

#### **Script action execution order**

By default, script actions are executed in the order as they appear in the script. But depending on the result of script actions and by special variable actions you can control the order in which script actions are executed. With these possibilities the script becomes a program with conditional jumps and better controller action execution. From this point of view, User Management Resource Administrator is a programming language to program the Windows network. To change the order in which actions are executed, labels are used for script actions. A label refers to a script action. To set a label for a script action, select the action in the script of the project (lower left part of the project window) and select menu option Actions, Set script action label. Each action can have only one label. When script execution continues at a label, the next action executed is the action with the specified label. If no action has the label specified, an error is generated and script execution is stopped. Note: since you can also jump to previous script actions, you can introduce deadlock situations where script execution never ends. It is the responsibility of the user to prevent this situation from happening. There are two general ways to change the order used to execute the script actions, described in the sections below.

#### **Script action execution order: Error handling**

When the execution of an script action fails, you have several options to control the execution of script actions. To specify these settings, select the action in the script of the project (lower left part of the project window) and select menu option **Actions-->Script action error handling**.

<b>Option</b>	<b>Description</b>
Continue with the script .	If this script actions encounters an error, it will (try to) continue normally with the next consecutive script action in the script



Jump to the script action with label xxxx	If this script actions encounters an error, it will jump to the script actions specified by the here specified destination label
Terminate the script but continue the session with the next line	If this script actions encounters an error the current script execution is terminated. The session continues by running the script with the next line of input.
Terminate the session	If this script actions encounters an error the current script execution is terminated and also the current session is terminated

**Script action execution order: Variable actions**

The *Script Action: Go to Label* on page 569 is used to continue script execution with another script action than the next script action. The referenced label can be a variable name.

**See also:**

*UMRA Basics* on page 3

*Getting started* on page 3

Help on Help

## 5.90. Lotus Notes Document Item Specification

In this dialog you can specify a Lotus Notes item that must be added to the list of items that will be collected for each document that is returned from the query.

In other words, you specify here the contents of a particular column in the resulting table.

**Item specification**

Specifies which item of the Document will be shown in the current column

**Name:** The name of the Item as know in Lotus Notes

**Type:** The data type of the resulting item in UMRA. Default is Auto. use a different value only if the auto conversion does not convert to the preferred type.

### **Error Handling**

Specifies what should be done if the specified item does not exist in the document.

- **Generate an error.** If the item does not exist in the document, the script action **Query document items** will report an error, and perform the general error actions that are configured. Use this if the existence of the item is required for further processing
- **Use an empty value.** Use this for instance if the item is not expected or required in all documents.
- **Use the value of variable.** If the item is not found in the document, the specified value contained in the variable will be added to the table instead. Use this if an empty value may cause confusion. This can be used for instance to show the text <not found> in the resulting table.
- **Error if conversion fails.** If it is not possible to convert the field to the specified data format, generate an error.

## **5.91. Lotus Notes Item Specification: General**

Specify here instructions how to create or modify a specific Document Item (field)

### **Item specification**

**Item Name.** The name of the Document Item to add. This is the name of the field as it is (or will be) known within Lotus Notes, for example "Fullname", or "ProxyAction"

### **Item type.**

Indicates the data type of the value associated with the item. The options available on the value tab reflect the choice specified here.

**Text:** The value of the item is a simple text string.

Text list: The value of the item is a list of text strings

date-time: The value of the item represents a date and time.

numeric: The value of the item is an integer number.

Notes reference: The value of the item is a reference to a different notes document.

### **Options**

Specifies what to do if there already is a field with the specified item name in the current document

Error if exist: The entire script action will not be performed and an error will be generated, if any of the fields with this setting already exist in the document

Delete existing first: If the current document already contains this field, the entire field is removed before the new field is added.

Append if exist: The new value will be merged with the existing value as specified in the value options. If there are no special options, the value will be appended at the end of the current one.

### **Item creation flags**

Several flags that determine specific notes setting regarding the field.

Sign: Items where this flag is set will be sealed when the document is signed, for instance with *Script Action: Sign document* on page 468.

Encrypted: Items where this flag is set will be encrypted, when the document itself is encrypted. Fields without this flag will not be encrypted.

Protected: Editor access is required to change the item.

Names: The item is a text field that contains a list of users or groups. often used together with the "readers" or "Authors" option.

Readers: The item is a item containing a list of readers, used for access control. The "names" option must also be specified if this option is specified.

Authors: The item is a item containing a list of authors, used for access control. the "names" option must also be specified if this option is specified.

Placeholder: The item is a placeholder field.

Summary: The item added to the document, and is also placed in the summary buffer of Lotus Notes. This is required for the item to be visible in any view. If the item is larger than 32 k it does not fit in the summary buffer and an error is generated. Use this setting if it is required that the particular field is always visible in views, or if you need to know that it cannot be shown. Your UMRA script may then react on the error situation either by making sure that the value is smaller than 32 k, add it without this flag, or perform some other required action. Only specify this flag if you really need to know if an item does not fit.

Auto-summary: The item is added to Lotus notes, and if it is smaller than 32 k it is also placed in the summary buffer which is required for it to be visible in any view. No error is reported if it is larger than 32 k. By default this is on. If you require a notification if an item does not fit, used the "summary" instead.

**See also**

**Script Action: Set items(s)**

## **5.92. Lotus Notes Settings dialog**

If you want to use UMRA to manage a Lotus Notes environment, you can here specify the general settings used by UMRA to connect to your Lotus Notes environment. Note that filling out this dialog is by itself not sufficient to configure UMRA for use with Lotus notes.

For detailed instructions how to set up UMRA for use with Lotus Notes see the *UMRA Lotus Notes user guide* on page 36

### **Important note**

If you reached this dialog by means of the tools/options menu, than all specified configurations are settings for the UMRA console application, and are used for all projects that are directly executed by the UMRA console.

If you reached this dialog by means of the "UMRA service/service properties" menu then all settings are used for scripts that are executed

by the Service. Make sure that all specified resources are specified so that the UMRA service has access to them.

#### **Enable Lotus Notes functions**

Check this box to instruct UMRA to initialize its connection with the Lotus notes environment using the below settings when the dialog is closed.

#### **Lotus Notes Settings**

##### **Lotus Notes Ini File**

The location of the Lotus Notes Ini file UMRA uses to setup the connection with the Notes environment. This can basically be a copy of the Notes.ini file which is used by the IBM Lotus notes client to connect to the Domino server. There are a few additions required to this file for proper operation. see the *UMRA Lotus Notes user guide* on page 36 for more information.

Most importantly, this file specifies the Notes security context (user.id) that UMRA uses to connect to the Lotus notes environment. For most purposes this should be a Notes user Id with administrative privileges.

##### **Password:**

Specify here the password for the Notes User ID file specified in the Lotus Notes .ini file.

### **5.93. Managing service projects**

As a rule, UMRA projects containing a script which takes its input from either a form (also known as Form projects) or another application (Automation projects), are stored on the server running the **UMRA Service**. To work with these projects, the **UMRA Console** application must be connected to the **UMRA Service**. See *UMRA Basics* on page 3 for general information on these topics and *Appendix B of the Getting Started Started* on page 69 for detailed instructions on installing the UMRA Service.

Once connected, you can manage the service projects on the **UMRA Service**. The following options are available:

**New**

Start a new service project. You will need to specify a name for the project before the project window is opened.

**Open**

Open the selected service project.

**Rename**

Rename an existing service project.

**Copy Project**

Copy the selected service project. The application proposes a new unique name for the copy of the project. Next, the new service project is opened.

**Delete**

Delete the selected service project. Note that when you select this option, the service projects are completely removed and you cannot undo this operation.

**Import**

Import service project(s) from file(s). You need to specify the file(s) that contains the service project you want to import. If you import a service project which already exists, you will be prompted to either replace the existing service project or create a new one. You can use the **Import** and **Export** options for restore and backup purposes.

**Export**

Export the selected service projects to a specified directory. You can use the **Import** and **Export** options for restore and backup purposes. Note that the files are stored in a single directory the folders do not represent subdirectories when exported. However, the folder names are stored in the project files themselves, so that on a subsequent import the project will be visible in the correct folder.

**Create folder**

Projects can be organized in a tree structure. Create folder creates a folder where you can store related projects. Note that the folder

structure is virtual, it does not reflect the layout of the actual project files on disk.

**Cut/paste**

Used to move selected project to a different folder

**Expand/Collapse**

Expand or collapses the selected folders

**Expand all**

Expand all folders

**Collapse all**

Collapse all folders

**Setup access**

Set or modify the access (execution) rights for all selected projects. This is convenient if you want to specify the same access rights for multiple projects at the same time

**Setup logging**

Set or modify project specific logging setting for multiple projects. For general logging options of the server, see the advanced tab of UMRA Service.

**Close**

Exit this window.

## 5.94. Name Generation Algorithms

User Management Resource Administrator supports the creation of unique user names automatically. This features is mainly used when creating user accounts in Active Directory or NT4 domains. In these environments, a user account has multiple names. Some of these names must be unique, e.g. no user accounts with the same names might exist. To generate these names automatically and to make sure they are unique, User Name uses name generation algorithms.

**Name Generation Algorithms**

A name generation algorithm is a set of rules that define how one or more names can be composed from other names and how the resulting names can be made unique. Example: when creating user accounts in

Active Directory 2 names must be unique. For this moment, we use the terminology Username and Full name for these names. For user accounts that are generated from a input file, the input data usually contains the First name, Middle name, Last name or a similar set of names. To generate the unique names, the name generation algorithm takes the three input names and according to the rules of the name generation algorithm it composes the 2 output names. If the names are not unique, the algorithm continues to iterate the generation cycle until the names are unique.



In User Management Resource Administrator, the number of input and output names, the methods used to convert the input names to output names and the way the names are made unique are completely configurable. All these configuration settings together are called a name generation algorithm. Name generation algorithms can be stored in files (.uga extension) and multiple name generation algorithms are shipped with User Management Resource Administrator. In most organizations, a policy is used how the user account names need to be composed. By choosing and perhaps customizing one of these algorithms you can let User Management Resource Administrator create unique names that adhere to your company's syntax requirements.

To choose an algorithm, select **Tools-->Options** from the main menu. Next, select the tab **Name generation** and click the **Manage** button:



The window shows example values for the input names and a list with available algorithms and the results of these algorithms according to the specified values for the input names. By specifying values for the **%FirstName%**, **%MiddleName%** and **%LastName%** variables you can see the results of each of the available name generation algorithm. In this window, you can **Add**, **Edit**, **Delete** and **Copy** algorithms. It is advised to copy algorithms first before you customize them.



In practice, User Management Resource Administrator uses variables to specify the input and output names of algorithms. So the input names of the algorithm are specified by passing variables. The results of the name generation algorithm is stored as a value for other variables. For more information on using variables, see *UMRA Basics* on page 3.

**See also:**

*Script Action: Create User (AD)* on page 3

*Script Action: Create User (no AD)* on page 68

*UMRA Basics* on page 3

## 5.95. Name Generation: Default input names

By default, the name generation algorithms can use the first, middle, and last name to compose the output name. To offer more flexibility, you can configure this. You can use any number of input variables and you can use any variable you like to compose output name. Further, for every input name, you can specify a sample value that shows up in the various dialogs and windows to help you configure the algorithms.

In the **Configure name generation settings** window you can configure the default input variables that can be selected from various dialogs and windows that are used to setup name generation algorithms. Use the **Add**, **Edit** and **Delete** buttons to manage the individual entries.

Note: If a variable is not part of this list, you can still use it in the name generation algorithms. The specification of these names is used only for displaying purposes.

**See also:**

*Name Generation Algorithms* on page 721

*UMRA Basics* on page 3

## 5.96. Name Generation: Embedded algorithms

Name generation algorithms are stored with the actions that use the algorithm. Examples: *Script Action: Create User (AD)* on page 3 and *Script Action: Create User (no AD)* on page 68. These actions have a property that specify the name generation algorithm. By configuring this property you can select and specify the name generation algorithm.

To manage name generation algorithms, you can export and import the algorithms using files. Normally these files have the .uga extension. When User Management Resource Administrator is installed, a number of default name generation algorithms are installed. To view and manage these algorithms, select menu option **Tools-->Options-->User name generation-->Manage**.

### See also:

*Name Generation Algorithms* on page 721

*Name Generation: Formatting functions* on page 724

*UMRA Basics* on page 3

## 5.97. Name Generation: Formatting functions

The formatting functions are used to change the value of an input name to a new value that is part of the output name. Each function converts a single input text to an output text. The formatting functions are executed in order. Each next function takes the result of the previous function as its input value. Some functions require additional arguments, some just operate on the input text.

### Test

Here you can specify a test input name. The result of the selected format function is shown in the field **Result**.

### Formatting functions

This is a list with all of the available formatting functions. When you select a function from the list, a description of the function is shown at the bottom of the window.

### Arguments

If a formatting function requires arguments, you can specify these in the **Arguments** section.

**See also:**

*Script Action: Format Variable Value* on page 549

*Name Generation Algorithms* on page 721

*UMRA Basics* on page 3

## 5.98. Name Generation: Iteration

The Iteration name part is used to make the output name of a *name generation algorithm method* on page 729 unique. The value of the iteration name part changes every iteration cycle. A simple example of a iteration name part is an increasing number, usually added at the end of the output name: 1,2,3,... . To specify the **iterator** field, open the *Configure name generation algorithm* on page 726 window and select a method in the lower section and click **Edit** or click **Add** to create a new method. The **Configure method of name generation algorithm** is shown. Click the **Iteration** button. The **Iteration name part** window is shown:



The window contains all options to specify the iteration name part. You can choose if the iteration name part must be an increasing number of an arbitrary sequence. For an increasing number, you can specify if the number of iterations should be limited. For arbitrary sequences, the number of iterations is limited to the number of entries in the sequence automatically.

If the output name uses an iteration name part, the iteration name part is always included, e.g. every time the method is accessed to generate a value for the output name. This holds even for the first time the method is called. For the first time, you might want to omit the iteration name part. This can be accomplished by using an empty value for iteration name part. To enable this feature, select the option **Always start the first iteration as an empty value**.

**See also:**

*Name Generation Algorithms* on page 721

*UMRA Basics* on page 3

## 5.99. Name Generation: Manage algorithms

A name generation algorithm generates one or more output names. For each output name, one or more name generation methods exist. A name generation method (short: method) in detail specifies how a single output name is generated from one or more input names. Further, the method specifies how to make the output name unique, e.g. iterate the method. The number of possible iterations with different outcome for the output name can be one, any other number, or unlimited. The most simple way to iterate the method is to add an increasing number at the end of the name: Jonh1, John2, John3, ... .

The purpose behind name generation methods is to support complete different ways (methods) of composing the output name if the first results are not unique. So the algorithm starts with the first method of an output name. If the result is not unique, it tries the next iteration of the same method. If the number of iterations is exhausted, the algorithm continues with the next method. Example: suppose an algorithm contains 2 methods to generate an output name. The first one has 5 iterations, and the second has an unlimited number of iterations. Then if no single name is unique, the algorithm generates the following possible names:

1. Method 1, Iteration 1
2. Method 1, Iteration 2
3. Method 1, Iteration 3
4. Method 1, Iteration 4
5. Method 1, Iteration 5
6. Method 2, Iteration 1
7. Method 2, Iteration 2
8. Method 2, Iteration 3
9. ...
10. ...

The methods 1 and 2 can use completely different rules to compose the resulting output names. To create a new method for a new algorithm, select **Tools--> Options** from the main menu. Next, select the **Name generation** tab and click the **Manage** button. Then click the **Add** button.



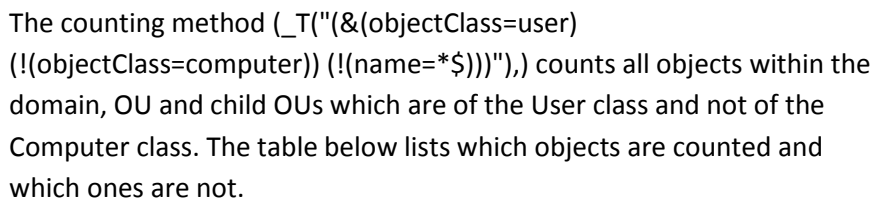
The **Configure name generation algorithm** window is used as a starting point to create a new or customize an existing algorithm. The window contains 2 lists: the upper list shows all of the output variables for the algorithm. If you select an output variable in the upper list, the lower list shows the methods configured to generate the selected output variable. For each method the name of the method is shown and the number of iterations supported by the method. The order of the methods shown corresponds with order used by the algorithm to generate the output name.

The most common operations initiated with this window are:

1. **Add a new output variable name:** Click the **Add** button in the upper section and specify the variable name of the new output name. If the variable name is not shown in the list, simply enter the name, enclosed in %-characters. The list only shows the variable names found in the active script properties. Once the output variable name is created, add one or more methods for the output name.
2. **Change the methods for an output variable name:** Select the variable output name in the upper list and select the method you want to change in the lower list. Click the **Edit** button in the lower section.
3. **Add a method to an output variable name:** Select the variable output name in the upper list and click the **Add** button in the lower section.
4. **Change the order of the methods for an output variable name:** Select the variable output name in the upper list and select the method you want to change the order for in the lower list. Use up and down buttons in the lower section to reposition the method.

- See also:**
- Name Generation Algorithms* on page 721
  - Name Generation: Formatting functions* on page 724
  - UMRA Basics* on page 3

If you right-click your domain in the network bar, you can choose the **Count users** command to count all the users in the specified domain.



Computer accounts, always ending on "\$", are not included in the count.

### 5.101. Network data

In this window you can specify how the network data should be displayed in project windows. When a project already contains project data there are three options:

1. A new project is started and the data are shown in the project window
2. The existing network data are overwritten
3. The network data are merged. This is only possible if the new and existing data are equal.

Activate the checkbox "Always ask this question when applicable" when you always want this option to be displayed in such cases.

### 5.102. Open UMRA project

An UMRA project either resides in a local file or on a server:

- **Local** - As a rule, UMRA projects with a project script taking its input data from a file (Mass projects), are stored locally.
- **Server** - As a rule, UMRA projects with a project script taking its data from a form selection (also known as Forms & delegation projects) or an external application (also known as Automation projects) are stored on the server running the **UMRA service**.

### 5.103. Name Generation: Setup algorithm methods

A name generation method (short: method) in detail specifies how a single output name is generated from one or more input names. For more information, see the topic *Name Generation: Setup algorithm methods* on page 729. To setup a method for a name generation algorithm you basically need to do three things:

1. **Specify the input names that must be contained somehow in the output name:** These name parts are specified as input variables. Note that the input names are not necessarily copied into the output name. Instead you can format these name parts. Example: suppose the output user name is composed of all characters of the last name and the first characters of the first and middle names. Then, the output variable **%Username%** can be composed from the input name **%LastName%**,

**%FirstName%** and **%MiddleName%**. Note that the order of the input names matters.

2. **Specify how to format each input name:** You can copy an input name directly into the output name, but you can also format the input name and copy the formatted result into the output name. Example: The output variable **%Username%** contains the first character of the variable that represents the first name: **%FirstName%**. The format function takes the first name as contained in the **%FirstName%** variable and converts it into the first character only: Jonh -> J. A number of formatting functions are available to change every name part. You can shorten the name, convert the case, remove and add characters, conditionally replace and delete characters and so on.
3. **Optional: Specify how to iterate the method:** One method can generate multiple names by using an iteration name part (iterator). The most simple iterator is an increasing number, added at the end of the output name: 1,2,3,... . Several options are available to specify the iteration sequence and to position the iterator in the final output name.

To setup the algorithm method, open the *Configure name generation algorithm* on page 726 window and select a method in the lower section and click **Edit** or click **Add** to create a new method. The **Configure method of name generation algorithm** window is shown:



In the upper section of the window, you can manage the name parts that compose the output name. You can add and delete name parts and add an Iteration name part. Further you can change the order of the name parts using the arrow buttons. In the lower section of the window, you can setup the format functions that apply to the name part selected in the upper list. Note that you cannot format the Iteration name part. To add a name part, click the **Add** button. The **Specify name part input variable** window is shown:



Select the input variable name from the list. If the list does not contain the variable name of your choice, simply enter the name in the field. You



can customize the input names shown in the list. See the topic *Name Generation: Default input names* on page 723 for more information. Once the input name has been selected, the corresponding sample value for the input name is shown. When you have finished, click **OK**.

When you setup the method, the result is shown at the bottom of the window. The **Temporary result name** shows the value of the output name according to the sample values of the input variables and current configuration of name parts and formatting functions. Again you can format this result by clicking the **Advanced** button. The final result is shown in the **Result name** field. If you do not specify any formatting functions in the **Advanced** section, both result names are equal.

**See also:**

*Name Generation Algorithms* on page 721

*Name Generation: Formatting functions* on page 724

Help on help

## 5.104. Password generation

For security reasons, User Management Resource Administrator support automatic password generation. For each user account that is created, a password can be generated automatically. You can configure the complexity of the generated passwords from simple (examples: sbjg, kyfd) to very strong (examples: 2v>`<J)G\0unOY, 3|}3aca9i>4H8Q{v`TS). User Management Resource Administrator further supports the password complexity rules as used in Microsoft Windows 2003/2000/NT networks. When the password is generated, it is stored in a variable. Next, this variable is used to actually set the password for the user account and export the password to an export file. As an alternative, you can also use no password, read the password from the input data or set the password to a constant value.

In the **Password generation** window, you can specify the rules used to generate a password. Note that this property can only be specified for the *Create User (AD)* on page 3 or *Create user (no AD)* on page 68 script action.

The **Password generator** window contains the following options:

### Predefined settings

The section **Predefined settings** contains a number of generation settings that are most easy to specify. Each setting specifies the value for the **Password generation** settings. These settings specify the minimum and maximum length of the password and the minimum and maximum number of characters of a specific type used to generate the password. Instead of selecting a predefined setting you can enter these values manually.

#### **Test password generation**

To see an example of a password generated according to the current settings, press the **Test** button.

#### **Output variable**

In the section **Output variable** you need to specify the name of the variable that must store the generated password. By default, this is the `%Password%` variable. It is advised not to change the name of this variable since it is used in related properties as well.

#### **More information:**

*Script Action: Create User (AD) on page 3*

*Script Action: Create User (no AD) on page 68*

*UMRA Basics on page 3*

[Help on help](#)

## **5.105. Script action property value**

Every script action in UMRA holds a number of script action properties which specify how the script action should be executed. In this window you can set the value for a specific script action property. There are three options:

1. **Value specified as a constant value:** Select option **Use the following value**. In this case, the value of the property is set to a fixed constant value. You can use this option only if the property

value must be the same each time the script is executed. This method is advised for fixed constant properties that have a value that is not used for other properties of the same or other script actions. Examples: the password flags (user cannot change password, password expired), the flag indicating if a share must be created for a home directory. These values are probably the same each time the script is executed.

2. **Value specified as a variable:** Select option **Use the following value**. With this option instead of specifying a value you specify the name of a variable. By default, the name of a variable should be enclosed in %-characters (e.g. %domain%). At run-time, the name of the variable is replaced by the value of the variable.
3. **Value not specified:** Select option **Do not specify a value for this property**. Some action properties are mandatory, others are optional. For optional properties, you do not always need to specify the property value. For instance, if you don't want to use it, you don't need to specify Active Directory attribute **Phone number** for a user account.

**See also:**

*UMRA Basics* on page 3

## 5.106. Scheduler

The Scheduler is used to schedule UMRA projects. The script of these projects will then be executed by the **UMRA Service** using the specified scheduling information. First of all however, you need to specify how the project script should be executed by the Scheduler. There are three options:

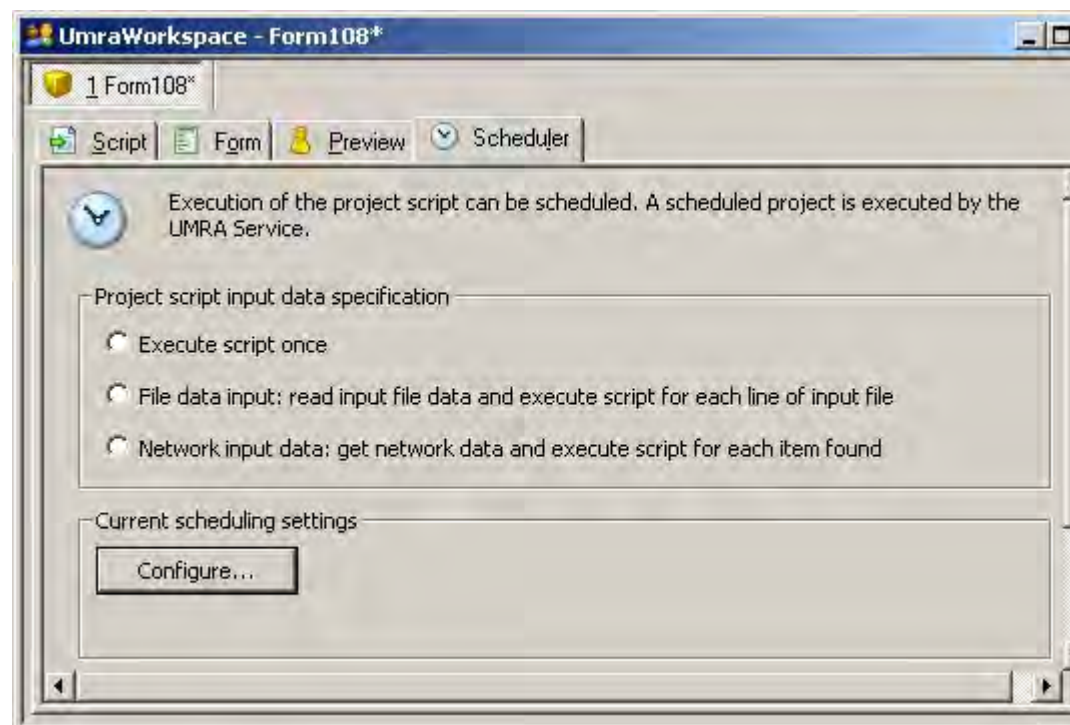
1. **Execute script once** - the project script is executed only once using the specified scheduling configuration.
2. **File data input** - the input file data (e.g. from a CSV file) are read and the **UMRA Service** executes the project script for each line of input data using the specified scheduling configuration.
3. **Network input data** - network data are retrieved and the **UMRA Service** executes the project script for each network item found, using the specified scheduling configuration.

### Scheduling configuration

Every possible *configuration* on page 735 can be specified, together with scheduling *exceptions* on page 739. These are time intervals which can either be included or excluded from the scheduling method.

When a project has been scheduled, it will appear under **Current scheduling settings**. In the example shown below for instance, a scheduling method has been defined which executes the project every Wednesday at 03:45.

It is also possible to test the scheduling specification by *previewing* on page 747 the first 10 scheduled times based on the selected scheduling configuration.



To define a new scheduling method or to edit an existing one, click the **Configure** button.

### See also:

*Setup scheduling* on page 735

*Setup scheduling - Exceptions on page 739*

*Setup scheduling - Adding an exception on page 745*

*Setup scheduling - Preview on page 747*

## 5.107. Set items

Specify here the Notes document to modify.

### Document Specification

Document variable:

The name of the variable that contains a reference to the document that must be edited. This variable may be obtained by using the action *Script Action: Get document* on page 454 prior to this action in the script.

### Document Items

The list with instructions for the creation or modification of the fields (a.k.a items) in the document. For each field to create/modify there is a separate entry in the list that specifies in detail how to create or modify the specific field.

Add: Specify a new field to create or modify.

Edit: Alter the specification for a particular field.

Delete: Delete the selected modification instruction.

## 5.108. Setup scheduling

Using the options in this window, a scheduling method can be configured.

### Scheduling method

- Not scheduled, never
- Every N seconds, where N is the specified number of seconds.

- Hourly, at N minutes, where N is the number of minutes (e.g. "0" is every whole hour, "30" is every half hour).
- Daily, at hh:mm where "hh" specifies the hour in two digits and "mm" the minutes.
- Once, at hh:mm dd/mm/yyyy. E.g. 7 July 2006 at 12.00 PM would be specified as "12:00 07/07/2006".

#### Include these days

Specify the days of the week to include in the schedule. You can either select individual days or use the buttons **All days**, **Weekends** and **Working days** to include every day of the week, the weekend, or all working days respectively.

#### Examples:

1. To execute a project every day, every 60 seconds:

**Setup scheduling**

Scheduling | Exceptions | Preview

Specify the scheduling parameters. Use the other tab windows to specify exception intervals and test the scheduling specification.

**Scheduling method**

- ☐ Not scheduled, never
- ☒ Every: 60 seconds (interval based)
- ☐ Hourly, at 30 minutes (specify minute 0 - 59)
- ☐ Daily, at 06:00 (HH:MM, HH=0..23, MM=0..59)
- ☐ Once, at 11/04/04/07/2006

**Options**

- ☐ Scheduling disabled

**Include these days**

- ☒ Monday
- ☒ Tuesday
- ☒ Wednesday
- ☒ Thursday
- ☒ Friday
- ☒ Saturday
- ☒ Sunday

All days  
Weekends  
Working days

OK Cancel Apply Help

2. To execute a project every (whole) hour every Monday, set the following options:

**Setup scheduling**

Scheduling | Exceptions | Preview

Specify the scheduling parameters. Use the other tab windows to specify exception intervals and test the scheduling specification.

**Scheduling method**

- ☐ Not scheduled, never
- ☐ Every  seconds (interval based)
- ☒ Hourly, at  minutes (specify minute 0 - 59)
- ☐ Daily, at  (HH:MM, HH=0..23, MM=0..59)
- ☐ Once, at

**Include these days**

- ☒ Monday
- ☐ Tuesday
- ☐ Wednesday
- ☐ Thursday
- ☐ Friday
- ☐ Saturday
- ☐ Sunday

**Options**

- ☐ Scheduling disabled

3. To execute a project every working day at 6.00 AM, set the following options:

**Setup scheduling**

Scheduling | Exceptions | Preview

Specify the scheduling parameters. Use the other tab windows to specify exception intervals and test the scheduling specification.

**Scheduling method**

- ☐ Not scheduled, never
- ☐ Every  seconds (interval based)
- ☐ Hourly, at  minutes (specify minute 0 - 59)
- ☒ Daily, at  (HH:MM, HH=0..23, MM=0...59)
- ☐ Once, at

**Options**

- ☐ Scheduling disabled

**Include these days**

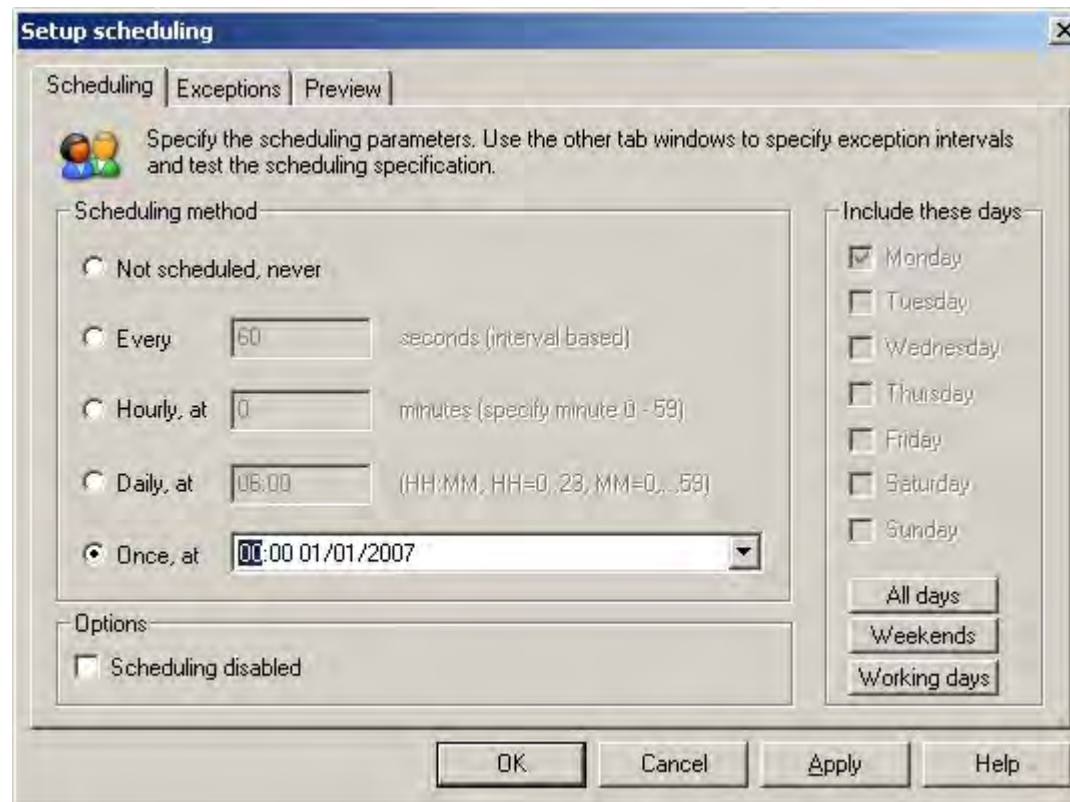
- ☒ Monday
- ☒ Tuesday
- ☒ Wednesday
- ☒ Thursday
- ☒ Friday
- ☐ Saturday
- ☐ Sunday

All days  
Weekends  
Working days

OK Cancel Apply Help



3. To execute a project once only on the 1st of January 2007 (midnight):



**See also:**

*Scheduler on page 733*

*Setup scheduling - Exceptions on page 739*

*Setup scheduling - Adding an exception on page 745*

*Setup scheduling - Preview on page 747*

## 5.109. Setup scheduling - Exceptions

### Exception intervals

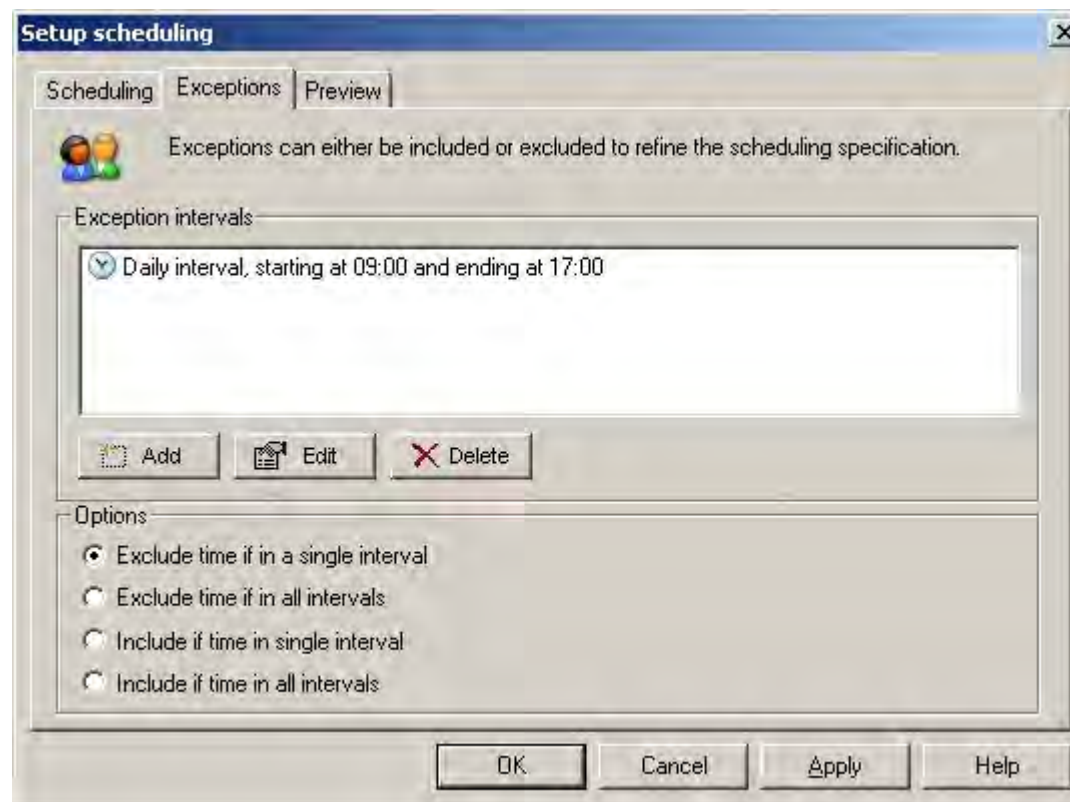
In addition to a scheduling method, you can specify exception intervals as part of a scheduling setup. This determines if a project should be

executed or not if the scheduled time is within the specified interval(s). If this time is excluded from the schedule, the project script is not executed. If it is included, the **UMRA Service** will execute the project script.

### Options

**Exclude time if in a single interval** - if the specified time falls into this repeat interval, it will be excluded from the schedule.

Consider a situation where you wish to execute a project every hour on every working day, but not during working hours (09:00 - 17:00). This can be achieved by setting the scheduling method to **Hourly, at 0 minutes** and including all working days. Next, an exception interval is defined which specifies that the time interval from 09:00 - 17:00 will be excluded from the schedule.



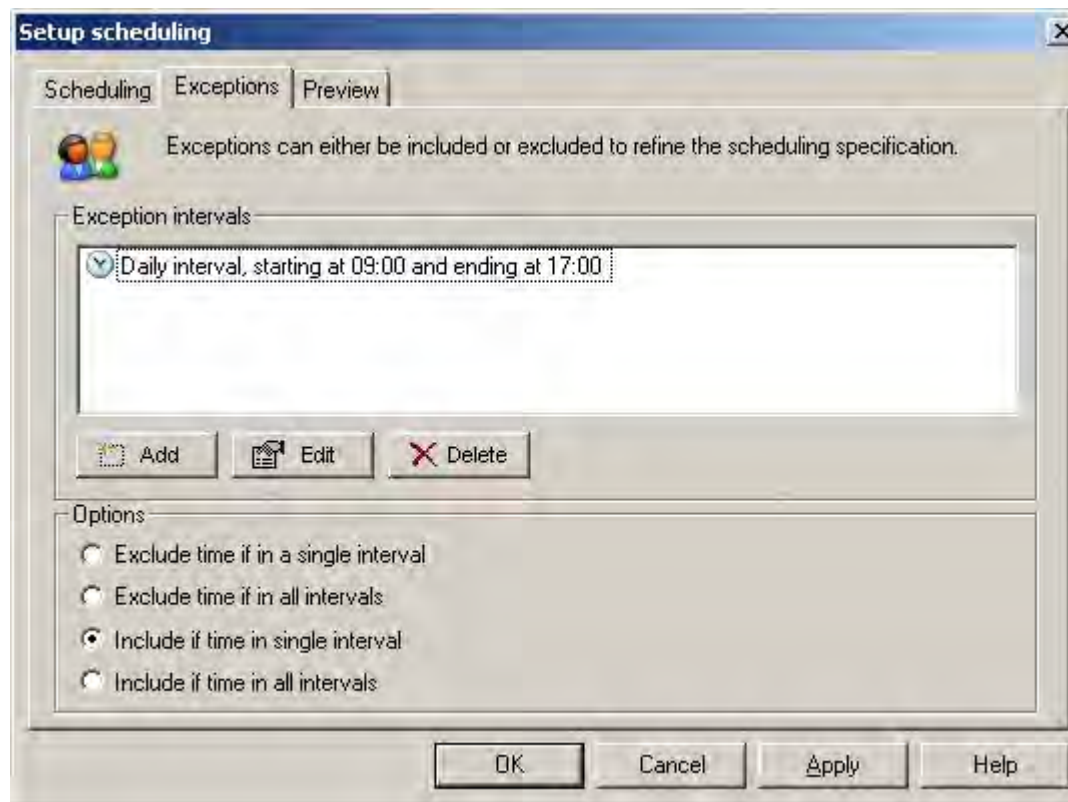
**Exclude time if in all intervals** - if the specified time falls into all of the defined repeat intervals, it will be excluded from the schedule.

Suppose now that you have a server A and B and that the project should not be executed during the time that BOTH servers are under maintenance. Server A is under maintenance from 08:00 - 11:00, server B from 09:00 - 11.30. If these intervals are specified in combination with the option **Exclude time if in all intervals**, the time period 09:00 -11.00 will be excluded from the schedule.



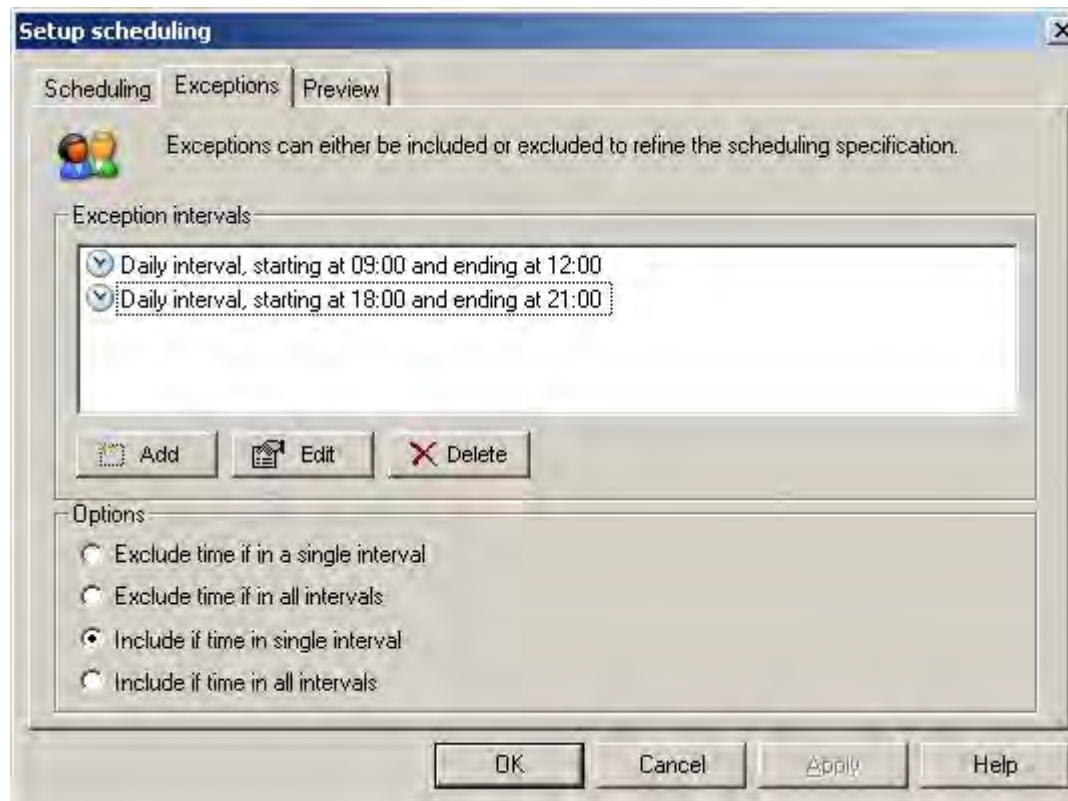
**Include if time in single interval** - if the specified time falls into (any of) the defined repeat interval(s), it will be included in the schedule.

For example, if you wish to run a project every three hours on Mondays from 09:00 to 17:00, you could set the scheduling method to **Every 10808 seconds** and include Monday. Next, an exception interval can be defined which specifies that the project will only be executed if the specified time is between the 09:00 AM and 17:00 PM interval:



Another example could be a situation where you wish to execute a project hourly every Monday from 09:00 - 12:00 and from 18:00 - 21:00.

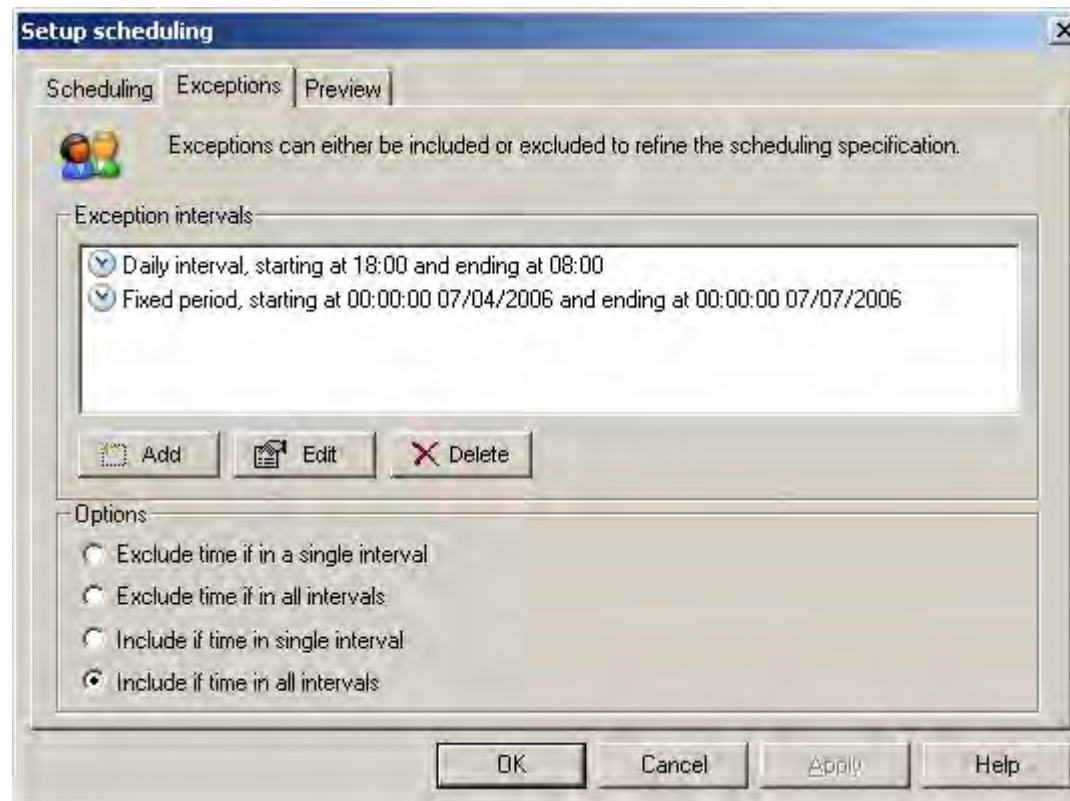
This can be achieved by scheduling the project **Hourly**, at **0** minutes on **Monday**. Next, an exception interval is specified which defines that the project is executed if the specified time falls in the 09:00 - 12:00 repeat interval OR in the 18:00 - 21:00 repeat interval.



**Include if time in all intervals** - the project is only executed when the scheduled time falls into all of the specified repeat intervals.

Consider a project which should only be executed during the time which all exception intervals have in common. The first exception interval runs from July 4th - July 7th, the second is a daily interval running from 18:00 - 08:00.

If the scheduling method has been configured to execute a project every hour every day, activating the option **Include if time in all intervals** using the above mentioned intervals will execute the project every hour from 18:00 PM on July 4th to 08:00 AM on July 7th.



**See also:**

*Scheduler on page 733*

*Setup scheduling on page 735*

*Setup scheduling - Adding an exception on page 745*

*Setup scheduling - Preview on page 747*

### 5.110. Setup scheduling - Adding an exception

In this window, an exception interval can be specified.

**Hourly repeated interval** - Here you can specify an hourly repeated exception interval between [First Minute] and [Last Minute] where **First Minute** and **Last Minute** can be any number between 0 and 59.

**Daily repeated interval** - Here you can specify a daily repeated exception interval between HH:MM and HH:MM. For example, you want a project to be executed every working day and every hour, except between 18:00 and 08:00.

**Scheduling exception**

Use this window to specify an exception interval. The interval can be an interval that is repeated every hour or every day or it can be a non repeating interval with fixed dates and times.

Exception interval specification

☐ Hourly repeated interval

First minute (included):  Last minute:

☒ Daily repeated interval

First time (HH:MM):  Last time:

☐ Non repeating interval

Start time:  End time:

**Non-repeating interval** - Here you can specify an exception interval as a single day or as a consecutive number of days, starting and ending at a specific time. To set the time, you can either enter the time directly in the **Start time** and **End time** fields or use the UP and DOWN arrows to select the required time.



This exception configuration can be used for instance, if the project should not be executed on Christmas Day or during your vacation.

In the example below, the non-repeating exception interval runs from 00:00:00 07/10/2006 until 00:00:00 07/21/2006.

**Scheduling exception**

Use this window to specify an exception interval. The interval can be an interval that is repeated every hour or every day or it can be a non repeating interval with fixed dates and times.

Exception interval specification

☐ Hourly repeated interval

First minute (included):  Last minute:

☐ Daily repeated interval

First time (HH:MM):  Last time:

☒ Non repeating interval

Start time:  End time:

**See also:**

*Scheduler* on page 733

*Setup scheduling* on page 735

*Setup scheduling - Exceptions* on page 739

*Setup scheduling - Preview* on page 747

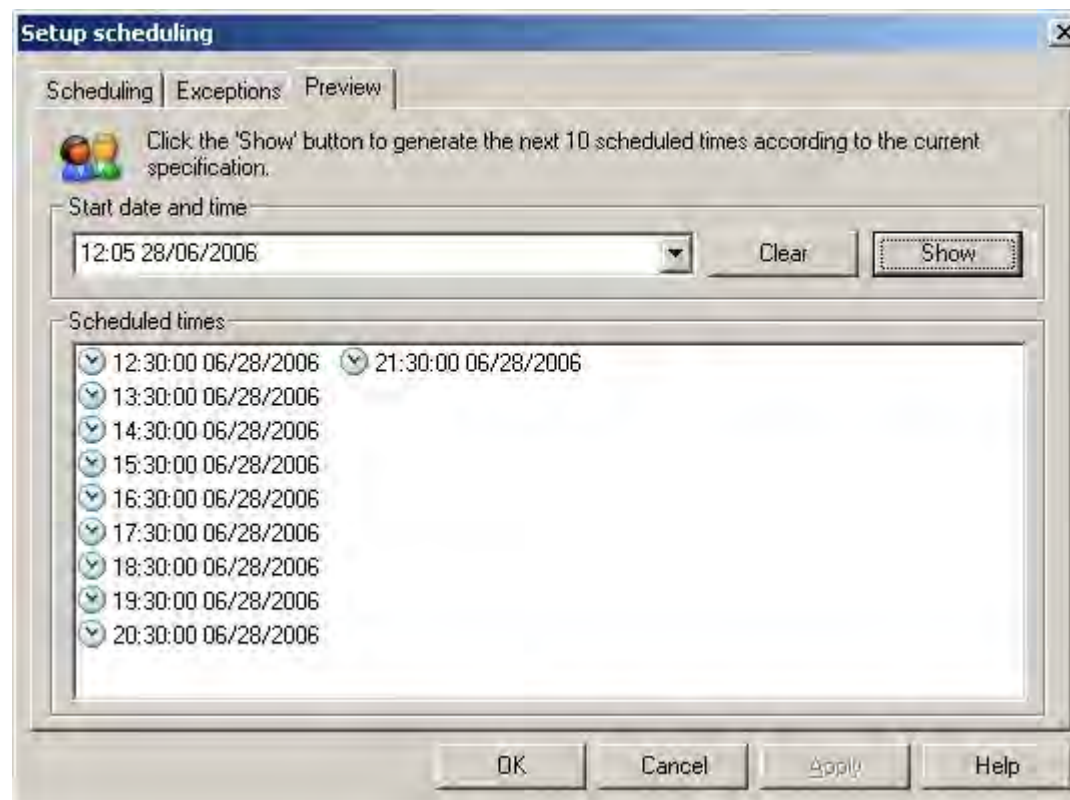


### 5.111. Setup scheduling - Preview

In this window, the next 10 scheduled times will be displayed according to the current scheduling specification when you hit the **Show** button. If the preview shows unexpected results, please check the scheduling parameters and exception intervals you have specified.

#### Example

If you have specified that the project should be executed every half hour at HH:30, the next 10 scheduled times may look as follows:



#### See also:

*Scheduler* on page 733

*Setup scheduling* on page 735

*Setup scheduling - Exceptions* on page 739

*Setup scheduling - Adding an exception* on page 745

### 5.112. Script action property value with yes/no option

This window is similar to the one described in *Script action property value* on page 748, but this specific property can also be specified by selecting **Yes** or **No**.

**See also:**

*UMRA Basics* on page 3

### 5.113. Script action property value output

The result of some script actions can be used by subsequent script actions. In User Management Resource Administrator, this is accomplished by using output properties. For these properties, the result corresponds with a value that is stored in a variable. This variable can then be used as a property value in subsequent script actions.

**See also:**

*UMRA Basics* on page 3

### 5.114. Script action property value - Output only

For this script action property, the value is determined automatically by UMRA. If necessary, this property value can be stored in a variable so that it can be used as input for other script actions.

### 5.115. Search and replace

With search and replace, you can search for text strings through all specified properties of the all the actions in the projects that are contained in the current workspace.

Its main purpose is to find where a specific variable or text is used in the relevant projects, and optionally replace the contents. For instance if you have specified a specific domain name at several places in the script, and you need to change it to an other one, you can use Search and Replace locate and rename all of the used names.

**Text Specification**

**Find What:** Specify here the text or variable to search for.

**Replace With:** Optionally, specify here the text to replace it with.

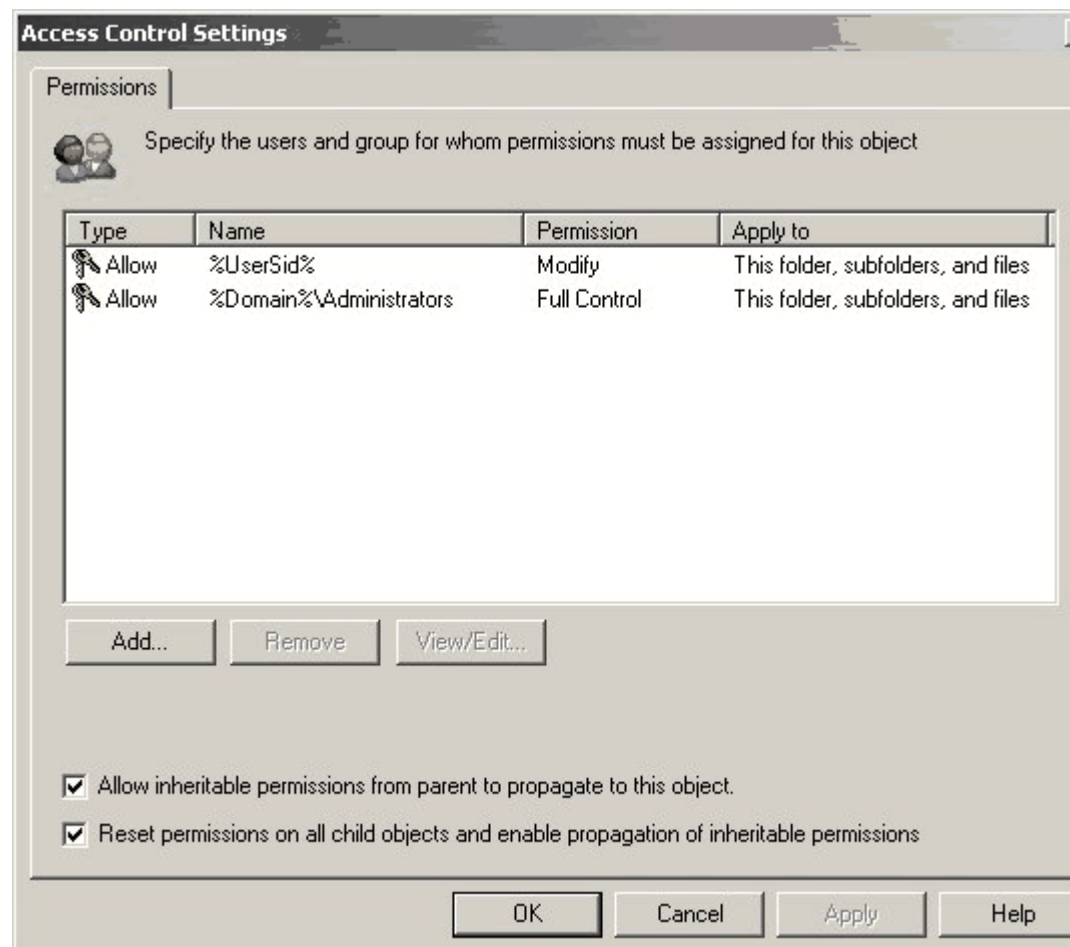
**Options**

**Match case:** Find only occurrences of text with the same case as the search string.

**Confirm replacement:** The program will ask to confirm each replacement.

### 5.116. Security - Access Control Settings

To specify detailed permissions settings you should use the **Access Control Settings** window. To start, select the action Create directory in the script section (lower left) of the project window. The properties of the script action are presented in the properties section (lower right) of the project window. Double click property Security or select the property and select menu option Actions, Properties of action property. Select option Use the following value and press the Edit button. The Directory security properties window is presented. Select a user account in the upper list or create a new account. Next, press the Advanced button. The Access Control Settings window is shown:



The window shows a list with all permissions setup for the account. For each permissions entry, a single line is shown. Use the Add, Remove, View/Edit buttons to manage individual permissions entries.

In Windows 2003/2000/NT, you can setup permissions that are inherited (copied) from the parent object. For directories, the parent object is the directory of which the target directory is a subdirectory. You can specify if inheritable permissions (as specified for the parent object), should be inherited by the target object. Use the option Allow inheritable permissions from parent to propagate to this object. If you do not select this option, the permissions of the target object are called protected since inheritable permissions from the parent object will not be copied to the target object.

**See also:**

*Security - Overview* on page 754

*Script Action: Create Directory* on page 341

*UMRA Basics* on page 3

[Help on help](#)

### 5.117. Security - Adding accounts and permissions

Access control settings are organized per account. For each account, permissions are specified. To start setting up a permission entry, you need to specify an account first. Then, you can set the permissions for the new account. To add an account, press the Add button in the Directory security properties window. See *Security - Overview* on page 754 for more information and how to access this window.



The **Specify input name** window is used to specify the name of an account for which permission settings are setup. The name can be specified as:

An existing account name of a user or group: Enter the name or press the Search button to search for the account name. Example: the administrators group of domain SEASONS, e.g. SEASONS/Administrators.

Note: at run time, User Management Resource Administrator converts the actual name into it's Windows 2003/2000/NT Security Identifier (SID). In order for this conversion to succeed, User Management Resource Administrator must be able to access a domain controller that maintains the specified account name.

A name containing a variable(s): In this case the variable name is resolved at runtime. This construction is often used for names containing 2 components with one well-known name. Examples: %Domain%\Administrators, %Domain%\Users. To select a variable, select it from the Variables list or enter the variable name and press Insert.

A single variable name: In this case, the name corresponds with the value of a single variable. The type of the variable can differ from the regular text type. For instance, when a user is created by User Management Resource Administrator in Active Directory, a specific object is created, which is the Security Identifier (SID) of the user account. This object uniquely identifies the new user account and where possible you should use the variable that holds this object. The object is by default stored in an output property variable - %UserSid% - and it should be used to identify the user account in subsequent script action properties. So if you create a user account in Active Directory and want to setup permissions for a directory that include the user account, use the variable %UserSid%.

**See also:**

*UMRA Basics* on page 3

[Help on help](#)

## **5.118. Security - Detailed permissions settings**

With the Permissions entry windows you can specify detailed permissions for an account. You can specify the permission settings itself and the objects to which the permissions apply. To access the window, open the Access Control Settings window first. Select an entry from the

list with permissions and press Edit. To add a new entry, press Add. You will be presented the Permission entry window.



The window contains several sections:

**Name**

Specify the name of the account for which you want to setup permissions. The account name can contain variables. For more information, see *Security - Adding accounts and permissions* on page 751

**Apply onto**

Specify the objects to which this permission entry specification applies by specifying one of the possible options. With this specification you determine if the specified permission applies to the target object (directory), contained child objects (files), contained subfolders or a combination of these options.

**Permissions**

A list with possible permissions. For each permissions you can either Allow, Deny or not specify the permission.

Apply these permissions to objects and/or containers within this container only

This option is almost never used.

**See also:**

*Security - Overview* on page 754

*Script Action: Create Directory*

[http://www.tools4ever.com/resources/manual/usermanagement6/script\\_action\\_create\\_directory.htm](http://www.tools4ever.com/resources/manual/usermanagement6/script_action_create_directory.htm) \t t4ehelppopup

*UMRA Basics* on page 3

Help on help

## 5.119. Security - Overview

User Management Resource Administrator supports Windows 2003/2000/NT permissions for all objects with security settings. For files and directories you can setup the specific security settings that should apply. User Management Resource Administrator uses similar windows as the Windows 2003/2000 graphical user interface to facilitate the configuration of the permission settings.

In User Management Resource Administrator, the security settings can contain variables. At runtime, these variables are replaced by their actual values to calculate and set the effective permissions. The security settings are primarily used for directories created with script action **Create directory** on page 341.



The window contains two lists containing names of accounts (upper list) and permission settings (lower list). The upper list shows the accounts for which permissions are defined for the target object (e.g. the directory). These accounts can be specified using existing account names or names containing a variable (As opposed to the equivalent Windows 2003/2000/NT window). By using variables, you can setup security settings for User Management Resource Administrator scripts, e.g. permissions for user accounts that do not already exist but are specified only by a variable name. To add new accounts, click **Add**. See *Directory security - Adding accounts and permissions* on page 751 for more information.



In the lower section of the window, you can setup the basic permissions for the account selected in the upper list. For the permission values shown, simply check the **Allow** or **Deny** option to configure the permission setting. In this permission list, you can setup only basic permissions. For most purposes, this will be sufficient. To setup more advanced permission settings, click the **Advanced** button.

In Windows 2003/2000/NT, you can setup permissions that are inherited (copied) from the parent object. For directories, the parent object is the directory of which the target directory is a subdirectory. You can specify if inheritable permissions (as specified for the parent object), should be inherited by the target object. Use the option **Allow inheritable permissions from parent** to propagate to this object. If you do not select this option, the permissions of the target object are called protected since inheritable permissions from the parent object will not be copied to the target object.

**See also:**

*Script Action: Create Directory* on page 341

[Help on help](#)

## 5.120. Security - Owner

For new directories and other items, you can specify the permissions and optionally the owner. See *Security - Overview* on page 754 for an introduction on this topic. To specify the owner, select the **Owner** tab in the **Directory security properties** window.

You can either specify the owner or let the operating system generate the owner for you. By explicitly specifying the owner of an item, you have more control. The owner can be specified using a fixed name or by using variables.

**See also:**

*Security - Overview* on page 754

*Script Action: Create Directory* on page 341

Help on help

### 5.121. Specify file input data

In this window you can specify how the text file should be read. The following options are available:

#### **File name**

This is the name of the text file that should be used as an input file.

#### **File type**

This specifies the delimiter for the individual fields.

#### **Text qualifier**

Specifies if the text is contained in double or single quotes. Select "none" if this is not applicable.

#### **Fixed width**

If the field width needs to be determined using specific character positions, these can be entered here.

#### **Additional settings**

**First line contains headers** - Activate this option if the first line of the text file contains the column headers.

**Insert row number column** - Activate this option if a column should be added containing row numbers.

**Field can contain multiple lines** - Activate this option if a field can be distributed over several lines.

#### **See also:**

*UMRA Basics* on page 3

*UMRA tables* on page 9

### 5.122. Specify group names

In this window the group name can be specified. A group name is specified using two different formats:

**Display** - This is the display name of the group. It is used only to refer to the group (e.g. Tools4ever\Users)

**LDAP** - This is the LDAP name of the group. This is the name which is actually used to identify the group in Active Directory.

**Specify group names**

Specify the group. The group is specified with 2 names. The first name is a display name with an easy readable format, for instance TOOLS4EVER\Users. The display name is used only to refer to the group. The second name is the LDAP name of the group. This is the name actually used to identify the group in Active Directory.

Group name specification

Display: Helpdesk Search

LDAP: LDAP://t4edoc.local/CN=Helpdesk,CN=Builtin,DC=t4edoc,DC=l

Variables: [dropdown] Insert

### 5.123. Specify input

This window is used to specify various different kinds of input for a script action. This can be a list of groups of which a user account should become a member, a list of e-mail addresses for a mail recipient, etc.

### 5.124. Specify input name

In the **Specify input name** window you can specify a name to be used as part of a configuration.

## CHAPTER

### 5.125. Specify new name for UMRA project

UMRA projects with a project script taking its input from a form or another application, are saved on the server running the **UMRA Service**. In the **Project name** field, you can specify a new name for your UMRA project.

### 5.126. Specify radio button text info

In this window you can specify the text which needs to be displayed for the radio button.

#### Display text

Here you can enter the text for the radio button

#### Variable value

You can either enter a fixed value to be associated with the radio button or a variable. When the user selects a radio button, the value associated with the selected radio button will be stored in a variable.

#### See also:

*Form fields - Radio button* on page 647

### 5.127. Specify variable info

In this window you can specify the name and value of a variable:

#### Variable

Select the variable from the list. The list shows the variables that are found in the script.

#### Description

The comment field describing the variable. From this description, the end user should understand what the variable is used for.

#### Value

The value for the variable or an instruction for the variable that is used. If the value is specified as a constant value, this value is shown in the corresponding field of the wizard.

**Example values**

Some possible values to help the end user to specify the value.

**5.128. Task scheduler overview settings****Update Specification**

*Update the task scheduler information every ... seconds.*

Specifies if the information in the overview window is refreshed automatically, and how often.

If the check box is selected, the information shown in the overview window is retrieved from the UMRA service automatically at the specified intervals. The default setting is once every 60 seconds.

If not selected, the overview window is not refreshed automatically. In both cases pressing F5 in the overview window will manually refresh the overview.

**5.129. Task scheduler overview window**

This table shows all projects that are known to the UMRA service, and for which scheduling settings have been specified. It also lets you configure several project settings with a right mouse context menu.

If a project is not in the table because no scheduling settings have yet been specified, and the project should be scheduled, open the project and choose Actions-->Scheduler from the main menu. Now an extra tab in the project named "Scheduler" is visible. After specifying the

configuration in this tab, save the project. Within one minute or less it will be visible in this window.

For each project in the table the columns mentioned below can be shown. Columns shown can be added or deleted by using the right mouse context menu that appears when clicking in the header of the table.

**Available Columns**  
**Enabled/Disabled**

Enabled: Project scripts will be executed at the times specified by its scheduling specification.

Disabled: The scheduling has been disabled, so the project will not be executed at the scheduled times.

**Task Type**

Currently there is only one type defined, in the future it is expected there will be more types.

- Scheduled Project- The project will be executed at the specified times.

**Description**

A description of the action performed at the scheduled time. Shows the name of the project.

**Execution Status**

- OK            The last time the script run it encountered no errors.
- Running     The project is currently being executed by the service.
- x Error      The last time the script run in encountered x errors.

**Last time**

The last time the project started.

**Next Time**

The next configured time the project will start. Note that this value has no meaning while the project is in the running state, as the next time is calculated only after the project has finished.

**Last duration**

The time it took the service to execute the project the previous time.

**Scheduling**

Short description of the chosen scheduling method (interval, daily, hourly, on-demand, etc).

**Scheduler info**

Some further details applicable to the chosen scheduling method.

**Actions available from this window**

Clicking with the right mouse button on a project, shows a context menu with several options pertaining to the project.

Open project

Opens the complete selected project, so it can be edited, usually chosen if you want to modify the script.

**Enable**

Directly enables the scheduling if it has been disabled.

**Disable**

Directly disables the scheduling. Current running projects will continue until finished, but will not be started again at the scheduled time.

**Run now**

Runs the specified project immediately.

**Refresh**

Refreshes immediately the display in the overview. By default it is refreshed once each minute.

**Delete**

Clears the scheduling settings of the project, and deletes the project from the Task scheduler overview window.

The project with its scripts are however not removed from the server, and can be opened again with file/open from the main console menu.

**View log**

Makes a copy of the project's associated eventlog file, and shows it in a window. When the window is closed the copy is removed.

**Overview properties**

Opens a window with the properties of the task overview window. Currently it has one option, to specify the frequency with which the window is refreshed with information from the service.

**Properties**

Opens the scheduling of the project, so these can be modified.



### 5.130. UMRA Console - Command Line Options

To support automatic execution of UMRA mass projects, the UMRA console application supports automatic startup command line options. When the UMRA console application is started using these options, the UMRA console application loads a projects when started and automatically starts the execution of the project. The command line options can be specified directly on the command line, or in a command line option file:

- UMgui.exe [options]
- UMgui.exe -commandfile=G:\UMRA\CommandFile.txt

In the command line option file, each line must contain a single option. Each option has a name. Some option have a value. Options can be specified using the following format.

- -option\_name=option\_value
- /option\_name:option\_value
- option\_name=option\_value
- -option\_name
- /option\_name
- option\_name

The following table shows the available options:

Option name	Option value	Example	Description
-AUTOSTART		-AUTOSTART	Option must be specified to enable automatic execution. If not specified, all other options are ignored.

-Project	File name of the project	project="G:\UMRA\CreateUser.upj"	The name of the mass project file that must be started. The project file can contain input data or the input data can be loaded from another file with option -inputfile.
-Inputfile	File name of the .csv, .txt file that contains the input data for the project	inputfile=G:\UMRA\Students.txt	The name of the file that contains the input data for the project. When not specified, the input data from the project is used.
-Separator	specification of input data separator characters	;;	Optional: the specification of input data separators. If not specified, the comma (,) - character is used.
-Textqualifier	Specification of the text qualifier used to read the input data from the input file.	" (double quote)	Optional: the specification of the input data text qualifier.

- IgnoreFirstLine		-IgnoreFirstLine	Optional: if specified, the first line of data of the input file is ignored.
-AutoQuit		-AutoQuit	Terminate the application automatically when ready.

### 5.131. UMRA Project Component - File data

The information an UMRA project script needs to work with, may come from a (CSV) file. This category is also known as a MASS project, since the project script will update all the resources included in the file.

For more information on using file data as input for an UMRA project script, see *UMRA Basics* on page 3.

### 5.132. UMRA Project Component - Form

The information an UMRA project script needs to work with, can come from a form. The end user can enter or select data in a simple form. These data can be linked to a variable which in turn can be used by the project script.

The form mentioned above is created by the administrator using the UMRA Console and delegated to a non-admin (e.g. Helpdesk employees). This category is therefore also known as Forms & Delegation.

For more information on UMRA projects, forms and variables, see *UMRA Basics* on page 3.

### 5.133. UMRA Project Component - Network data

The information an UMRA project script needs to work with, may come from a network selection. Network resources can be added to the project by activating the Actions-Network-Form fields window. Right-

click the required network resource and choose the data you wish to add:

- All users
- All user details
- Local groups
- Global groups
- Domain controllers
- Servers
- Workstations

For more information on using data as input for an UMRA project script, see *UMRA Basics* on page 3.

#### **5.134. UMRA Project Component - Preview**

If an UMRA project includes a form, the form-layout can be previewed. By default, changes in the form will be instantly reflected in the form preview.

#### **5.135. UMRA Project Component - Script**

Each and every UMRA project contains at least a project script. An UMRA script is used to perform a specific user account or network management task. This can be the creation of a user account for instance, together with a home directory, home share, group memberships, etc. for the new user.

In addition, you can specify the input data the project script needs to work with. This can be file data, network data, data from a form selection or external application data.

For more information on UMRA project scripts, see *UMRA Basics* on page 3.

To get context help on a particular script action, select the script action in the general actions tree, and press F1

### 5.136. UMRA Project Properties - Description

In the **Project description** field you can enter a description of the current project. Since UMRA projects can become very complex depending on the solution you want to build, it is strongly advised to fill this in. Especially when you are working with multiple projects, it will make it

## CHAPTER

easier to manage your projects.

### 5.137. UMRA Project Properties - Form Fonts

In a form, you can use multiple fonts. For all form fields containing text, the font can be configured. UMRA works with font styles. A number of predefined font styles are used to draw all of the text. For each applicable form field, you can specify the font style that must be used for the form field text.

In UMRA, a form can have its own specification of all font styles or it can use the global font styles.

### 5.138. UMRA Project Properties - Form options

In this window you can set some form control options.

#### Initial project

An initial project is specified for all form projects where the script of another project needs to be executed before the form is shown for which you have specified an initial project. This concept is shown in the figure below. The initial project contains a script action of which the output is stored in a table variable. In the second project, project B, the content of this variable is displayed in a generic table. This is only possible when the script of project A is executed before showing project B. Project A is therefore defined as an initial project for Project B.



**Project name**

The name of the UMRA project which needs to be set as an initial project.

**UMRA Forms client****Show form project in Available forms window of UMRA Forms.**

If this option is checked (default) the form is shown in the UMRA Forms application if the UMRA Forms user has sufficient access rights to use and run the form. This option is used when multiple forms together are used as a wizard. In this scenario, the UMRA Forms end-user should be able to select the first form of the wizard from the list shown in UMRA Forms. In that case the other forms of the wizard will not be displayed in the list.

**Popup message options****Status information stored in the %ScriptMessage% variable.**

Select this option if a popup message must be shown in the UMRA Forms application when a form is submitted. Note that the project script must set this variable (example: %ScriptMessage% = Creating user account %UserName%). If the variable is not set, no popup message will be shown.

**Error message**

Select this option if a popup message must be shown if an error occurs upon submitting a form.

**Preview update option****Update form preview when preview window is activated.**

Deselect this checkbox if you do not wish to have the form content constantly updated during the design phase.

**5.139. UMRA Project Properties - Format**

In this window you can specify the display characteristics for the form. Note that these settings only apply to the overall form. The display settings for the individual form fields have to be specified in the display tab of the relevant form field.

**Left margin**

Left margin, specified in pixels. Default is "10".

**Right margin**

Left margin, specified in pixels. Default is "10".

**Top margin**

Top margin, specified in pixels. Default is "10".

**Vertical spacing**

This is the vertical space between the individual form fields of the form, specified in pixels.

**Background color**

Background colour for the form. By default, the background colour is white. Click the **Edit** button if you want to change this or if you want to specify a custom colour.

## 5.140. UMRA Project Properties - General

In this window a summary is given of all project information.

**Project identification**

These options are all related to how the project is identified:

**Project name**

Name of the project.

**Display name**

Display name of the project

**Project ID**

A unique identification number generated by the UMRA service. The Project file shows the name of the file that stores the form project for local projects. For server projects, the name includes the name of the UMRA server.

**Project location**

The project location field indicates where the project is located. There are two possibilities:

**Server project maintained on server**

By default, projects with a project script taking its input from a form or an external application are stored on the server running the **UMRA Service**. The name of this server will be displayed in this field (e.g. "Server project, maintained by UMRA Server Server-A")

**Local projects**

By default, projects with a project script taking its input from a input data file are stored locally. The path to the local file is displayed in this field (e.g. C:\Program Files\Tools4ever\User Management Resource Administrator\Projects\CreateUsers.uwp).

Note that it is possible to change a local project to a server project and vice versa by clicking the **Change** button. In that case you will run the risk that the project needs to be (partly) reconfigured. A warning to this effect will appear and you will be asked for a confirmation.

**5.141. UMRA workspace**

An UMRA workspace is a collection of one or more UMRA projects. An UMRA project is always part of a workspace. Projects can be added to an existing workspace or you can create a new workspace.

When you create a new Mass, Form, or Automation project, UMRA will automatically create a workspace for you. The name of this workspace will initially be UMRAWorkspace.uws, but this can be changed by choosing the **Save workspace as** command from the **File** menu. Please note that a workspace project only contains references to UMRA projects.

For more information on UMRA's key concepts and architecture, see *UMRA Basics* on page 3.

**5.142. UMRA Project Properties - Initial variables**

Initial variables are usually specified for a Form Wizard (i.e. a solution consisting of multiple form projects). In such a scenario, you want to make sure that variables which are used in the project script, have an empty value **when the project script is executed for the first time**.



### 5.143. UMRA Project Properties - Network data

Network data can be specified as input for the project script. If network data have been specified, the **Network data description** field indicates where these network data originate from.

### 5.144. UMRA Project Properties - Options

**Script execution time-out value (seconds).**

Script execution always continues, regardless of whether a time-out has been specified or not. This option is to avoid a situation for instance, where script execution goes into an eternal loop because of errors in the script. When the script does not finish within the specified time-out period, script execution will continue, but an error is generated and the script result variables are not returned.

### 5.145. UMRA Project Properties - Security

When an UMRA project is maintained by the UMRA service you can specify the user accounts that are allowed to execute the form and project script.

For UMRA, there can be three different kinds of user accounts:

User account with	Description
Full control	These user have access to push forms to the UMRA service, setup, delete, manage all forms, project scripts and security settings. The number of user accounts with this type of access should obviously be very limited.

Form access only	These users can see and submit a form. When such a user connects to the UMRA service using the UMRA Forms client, the form is presented to them. The user can then specify the various fields of the form and let the UMRA service execute the script of the form project. The accounts can be configured for each individual form.
No access	These users can connect to the UMRA service but no projects will be shown.

## 5.146. UMRA service - Advanced options

### UMRA service network data cache

The UMRA Service maintains a cache with network data. When a UMRA form project contains network data, for instance a table with user accounts and the UMRA Service is requested to produce the contents of a form, the UMRA Service accesses the cache to find the data.

The purpose of the cache is to minimize network traffic. By using the cache with network data, the amount of network traffic caused by the UMRA Service is greatly reduced. The drawback is that the data is less accurate. Example: if the user account is deleted with another application, the deleted user account might show up in an UMRA form because the cache is not up-to-date.

The data in the cache is valid for a certain period of time. Once expired, the data is refreshed. The refresh period is specify in seconds. The default is 900 seconds (15 minutes).

An additional option is available to empty the cache completely. When selected, the cache is automatically rebuilt when form data is requested from the UMRA Service.

### User specific log files

The UMRA service can be configured to use user account specific log files. That way, when a user executes a form project with the UMRA forms client, the log messages of his project are logged in a user-specific log file, instead of in the general log file. All files are stored in the "Log" subdirectory of the UMRA service.

### 5.147. UMRA service - license

A UMRA service license is enabled by configuring one or more license codes for the UMRA service. A license code for the UMRA service can be obtained from your UMRA reseller. To install the UMRA license code, connect to the **UMRA service**. Select menu option **UMRA Service, Connect...** When connected, select **UMRA Service, Service properties...** and select the **Service License** tab.

In the **Configure service - Service license** window you can:

1. **Configure** UMRA service licenses - Add and delete license codes to the UMRA service.
2. **Copy to service** - Copy the service license codes installed for the UMRA console application to the UMRA service. You should only do this, if the installed license code enable the service functions.
3. **Console** - View and configure the UMRA console licenses. This option is available to manage and setup the UMRA console licenses. You can choose the main menu option Help, License... to setup the UMRA console licenses.

When the **UMRA service** is installed for the first time, a demo license is installed automatically. The demo license will run for 30 days. For more information on UMRA licensing, see *License model* on page 707.

### 5.148. UMRA Service - service access

A limited number of users should have full control to manage the UMRA service. These user accounts have full access to:

- configure the UMRA service
- add, manage and delete all form projects maintained by the UMRA service
- setup the security for each form project maintained by the UMRA service
- setup the security for the UMRA service itself

In the **User and groups allowed to manage the service** window, the users and groups are listed which are currently allowed to manage the UMRA Service.

**Adding users**

Click the **Add** button if you wish to add a user or group to the current list

**Editing users**

If you wish to edit an entry, then select the user or group you wish to change and click the **Edit** button.

**Deleting users**

If you wish to remove a user or group from the list, then click the **Delete** button.

**5.149. UMRA service deletion - Delete all files**

When the UMRA Service is running, the service will create a number of files directories. For instance, all logging information and UMRA form projects are stored in files. When the service is deleted, you have two options:

1. Delete all files found in the UMRA Service directory and the directory itself: This will delete all the files originally installed when the service was installed and all files created in the UMRA Service directory.
2. Only delete the files that were originally installed.

**5.150. UMRA service installation - Admin group**

The UMRA service must have sufficient administrative privileges to execute its tasks. These privileges are determined by the service account used by the UMRA service. By adding the service account to one or more administrative groups, the account can be granted sufficient access rights.

**5.151. UMRA service installation - Server**

The UMRA service is setup from with the UMRA console application using a wizard. To start, select menu option UMRA Service, Install or

Upgrade service...Select option Install or upgrade the service and press Next.

Enter the name of the server on which you want to install the UMRA service. This can be any computer running Windows 2003/2000. It is recommended to install the UMRA service on a server that is close to a domain controller or on the domain controller itself.

Click Next to continue.

### **5.152. UMRA service installation - Port**

The UMRA service communicates with the UMRA console and UMRA forms client using the TCP/IP protocol. For this communication a port must be specified. By default, port number 56814 is used by the UMRA service to communicate but you can specify any other port.

When starting up the communication with the UMRA service, the UMRA console and UMRA forms applications will try to connect using the default port number first. If this fails, the port can be configured differently. It is recommended to use the default port.

### **5.153. UMRA service installation - Service account**

The UMRA service uses an Windows 2003/2000 account to run. If the account does not exist already, it is created by the UMRA service installation wizard. It is recommended to use an Active Directory domain account, not a server local account.

All scripts executed by the UMRA service are executed by this account with respect to the Active Directory - Windows 2003/2000 security settings. Therefore, this account must have sufficient administrative privileges. In the *next step of the wizard* on page 774, the group can be added to an administrative group.

By default, the UMRA service installation wizard specifies the following name for the service account: DOMAIN\UmraSvcAccount. Further, a random strong password is generated. This password is not known to anyone. If the account does not exist, the account is created with the generated password.

If the account does already exist, the password will be incorrect. In this case, you must specify the correct password or change the name of the service account to a non-existing name.

### 5.154. UMRA service installation - Service directory

Before the UMRA service is created on the remote machine, the UMRA service executable files are copied to the computer in a particular directory. The UMRA service installation wizard automatically determines the target directory but you can specify any other directory.

Note that the name of the directory is specified relative to the target computer, e.g. G:\UMRAService means the the directory UMRAService on the local drive G:\ **of the computer on which the UMRA service is installed.**

### 5.155. Variable generic table

In UMRA, there are several script actions for which the output variable is stored in table format. The **Variable generic table** window is used to specify the name of the variable and to define the columns of the table

Note that a table variable only holds the data of the table, not the names of columns. You therefore need to add the names of the columns that can be shown with the generic table.

#### Specifying columns for table type variable

The special table type **Variable** takes its input from a variable containing table data. This variable (which is the the result of script actions such as *Generate Generic table* on page 527, *List services status* on page 373, *Get User Table*) on page 51 does not contain any header info. This means that you need to specify the correct column names. For variables which are the result of one of the before mentioned script actions, this is simply a matter of selecting the right column template.

Template	Description
4 columns	4 columns (Column1, Column2, etc.) to hold data.
8 columns	8 columns (Column1, Column2, etc.) to hold data

12 columns	12 columns (Column1, Column2, etc.) to hold data
User info	To be used in combination with the <i>Get user table</i> on page 51 script action. It will set the columns for the %UsersTable% variable.
Services status (without config info)	To be used in combination with the <i>List services status</i> on page 373 script action. It will set the available columns for the %ServicesTable% variable, but without the columns start up type (text), start up type (code), binary file and log on as.
Services status (with config info)	To be used in combination with the <i>List services status</i> on page 373 script action. It will set the available columns for the %ServicesTable% variable, including the columns start up type (text), start up type (code), binary file and log on as.
Printer documents	To be used in combination with the <i>List printer documents</i> on page 380 script action. It will set the available columns for the %DocumentsTable% variable
List files and or directories	To be used in combination with the <i>List files and/or directories</i> on page 367 script action. The name of the output variable is user defined.

When you have selected the required template and click the **Set columns** button, the columns for the specified template will be added.



**See also:**

*UMRA tables* on page 9

*Script action: List services status* on page 373

*Script Action: List printer documents* on page 380

*Script Action: Get user table (AD)* on page 51

## C H A P T E R

### **5.156. Variable list**

In this window, you can define a list of variables to be used for testing purposes. A variable is defined by its name and value. These testing variables are resolved at runtime.

Click the **Add** button to add a test variable name and its corresponding value.

## **6. No help available**

There is no help available for this window.



## 7. Index

### A

Access and Security • 12

Access rights are not propagated to Active Directory • 10, 11

Access rights of mailboxes when using Out Of Office actions • 6, 10

Accessing Exchange 2010 functionality from an UMRA project • 1

Action Property script phrase • 30

ActionProperties • 19

Active Directory • 3, 66

Active Directory permissions • 308

Active Directory utility • 321

Add directory service object • 22

Adding a directory service object • 58, 61

Adding the person directory service item • 44, 46

AddRow • 18

Administration Requests database • 39

Advanced Settings - domains • 325

Advanced Settings - general settings • 325

AFAS Online • 76

Agent service session • 321

Allow access on the whole organization • 13

Allow access per mailbox user • 13, 15

AppendColumn • 18

Appendix A - Script actions • 4, 10, 26, 54

Appendix B - Installing the UMRA Service • 31, 57, 392

Application input data • 16

Aura • 10, 258

Aura connector installation • 88, 258

Aura installation 1 - Architecture • 89

Aura installation 10 - Test the UMRA-Aura-WebService • 99

Aura installation 2 - Prerequisites • 89

Aura installation 3 - Create user account • 89, 96

Aura installation 4 - Create web site • 90

Aura installation 5 - IIS / ASP.NET 1.1.4322 • 92

Aura installation 6 - Aura license file • 94

Aura installation 7 - Aura data access • 95

Aura installation 8 - Update web.config • 96, 97

Aura installation 9 - Test the web-site • 97

Auxiliary project - Print job list - HP\_1220C • 2

### B

Background information about access rights on mailboxes • 10, 11, 16

Basic section • 15

Boolean Value Dependent Action Property script phrase • 35

Built-in variables • 18, 19, 33, 34, 329

### C

ClearVariables • 5

Component Object Model (COM) • 1

Concept • 1

Condition criteria - Setup • 330

Condition criteria - Setup criterion • 330

Conditional Action Property script phrase • 32

Configuration and settings • 12

Configuration of the Umra Service • 82

Configuration section • 12, 18, 51, 321, 322

Configure predefined variables • 331

Configuring a secure web-site with IIS • 54, 90

Configuring the UMRA console for use with Lotus Notes • 20, 31, 35, 39

Configuring the UMRA project • 23, 41

Configuring the UMRA service for use with Lotus Notes • 20, 31, 35

Connect • 5

Contact • 158

Contacts • 20, 30

Context sensitive Help • 322

Control running UMRA service projects • 331

Create replica • 39, 41, 43

CreateTable • 18

CreateTable2 • 18

CreateTable3 • 19

Creating a directory service item • 11

Creating a Forms example project - Reset password • 5, 26

Creating a MASS example project - Mass create users • 2, 10

Creating directory service items with OpenLDAP on Linux • 39, 44

Creating the website • 36

Creating user accounts in Microsoft Active Directory using LDAP • 49, 58

Creating user accounts in Novell eDirectory • 19

Customizing name generation algorithms • 4

## D

Data specification - Text list • 49, 331

Database • 282

Database query - Database specification • 332, 333, 363

Database query - Query • 332, 333, 363

Database setup - MS-Access (Jet) • 332, 333

Database setup - Other databases • 27, 332, 334

Declaration • 15

Delegating user account management tasks - Forms module • 3, 26

Delete Options • 324

Deleting a directory service item • 12

Deleting user accounts in Novell eDirectory • 32

Deny 'Receive As' access for 'Domain Admins' on mailboxes • 10, 13

Directory Service tasks • 11

Distribution group • 160

Domain Controller Options • 324

Dynamic actions library • 39

## E

Education • 88, 258

Encrypted properties • 17, 21, 38

Example • 41

Example 1 - Creating an LDAP table showing all disabled users in a domain • 19

Example 2 - Creating a form table to connect to a database • 21

Example 3 - Creating a variable with table data and showing the content in a form table • 26

Exchange • 88

Exchange 2000/2003 • 88

Exchange 2000/2003 requirements • 9

Exchange 2007 • 45, 101

Exchange 2007 Client Access Role • 6, 9

Exchange 2010 • 20

Exchange server • 166

Exchange Web Services certificate • 6, 7

ExecuteProjectScript • 6

Expiration date • 336

## F

File input data • 10, 1

File system • 319

File System • 171

Fixed table • 10

For each - Input variables • 304, 305, 337

Form action - Check form input • 25, 337

Form action - Execute command line at client workstation • 10, 340

Form action - Execute script of form • 338

Form action - General • 338, 341, 342, 343, 356

Form action - Iteratively execute project script • 33, 339

Form action - Return current form • 340, 354

Form action - Return other form • 33, 340

Form action - Set variable value • 340, 342, 355

Form fields - Button • 341, 346

Form fields - Checkbox • 27, 341, 346, 355

Form fields - Display • 343, 348

Form fields - Input text • 344, 355

Form fields - Name • 340, 345

Form fields - Picture • 345, 356

Form fields - Radio button • 346, 356, 418

Form fields - Static text • 347, 355

Form fields - Table - Columns • 347

Form fields - Table - Data refresh • 349, 352

Form fields - Table - Exclusions • 350

Form fields - Table - Fixed data • 350, 351, 361

Form fields - Table - Generic table • 351, 363, 374, 377, 379, 381

Form fields - Table - Network call parameters • 351, 352, 353

Form fields - Table - Network table • 349, 351, 352, 361

Form fields - Table - Options • 25, 339, 350, 353

Form fields - Table - Row icon image • 354

Form fields - Table - Type • 349, 351, 352, 353, 354, 355

Form fields - Vertical space • 348

Form input data • 12, 1

Form layout • 13

Form project - Form fields • 355

Form projects • 1

Form table return variable • 7

Form table types • 4

Formatting tables • 18

Function modules • 356, 364, 383, 384, 385, 386

## G

General • 2

General section • 15

General user Actions • 54

Generate Generic table - Script • 16

Generating user names • 1

Generic table - Column names • 364

Generic table - Database query • 10

Generic table - Introduction • 361

Generic table - LDAP query • 6

Generic table - Run test • 361, 376, 378

Generic table - Table type • 361, 362

Generic table - Variable • 364, 378

GetCellText • 17, 19

GetCellTextEx • 20

GetColumnCount • 20

GetColumnName • 20

GetConnectionInfo • 6

GetConnectionString • 7

GetFormTable • 16

GetHostName • 7

GetLogMsg • 8

GetLogMsgCount • 8

GetLogMsgEx • 8

GetPortNumber • 9

GetRowCount • 21

GetScriptExecutionInfo • 10

Getting Started • 3, 386, 387, 389

GetVariableDataTable • 11

GetVariableInfo • 10

GetVariableText • 11

GetVersion • 12

Goal • 15, 79

Google - Action

    Google Setup Connection • 70

Google - Connections • 71

Google - Registry settings • 72

Google - Requirements • 70

Group management • 317

## H

HideVariable • 10, 12

## I

ID Vault • 12, 43, 44

IDD\_DIALOG\_CYCOS\_CUSTOMFIELD\_OUTPUT • 326

IDD\_TAB\_ACTIONITEM\_CYCOS\_GET\_ATTACHMENT • 326

IDD\_TAB\_ACTIONITEM\_CYCOS\_GET\_CUSTOM • 326

IDD\_TAB\_ACTIONITEM\_CYCOS\_SET\_CUSTOM • 326

IDD\_TAB\_ACTIONITEM\_LN\_ACL-forwarded • 326

IDD\_TAB\_ACTIONITEM\_LN\_QUERY\_ITEMS-forwarded • 326

IIS configuration Windows 2003 • 34

IIS configuration Windows Server 2008 • 49

Initial project specification • 13

Install/upgrade software • 323

Installation • 1, 2

Installation and upgrade wizard - Specify the target domain • 323

Installation and upgrade wizard - Specify the target domain controller • 323

Installation and upgrade wizard- Installation and upgrade options • 323

Installation of the Notification Package • 80

Installing the UMRA program files • 8

Installing UMRA • 8

Installing UMRA PSM for the first time • 79

Integrate UMRA with other applications using COM • 63, 386

Integrating Active Directory in existing (Sharepoint) web portals • 6

Interface modules • 361, 364, 383, 384, 385, 386

Introduction • 3, 26, 3, 23, 33, 1, 13, 39, 49, 1

Introduction Exchange 2007 • 19, 1

Introduction Exchange 2010 • 8, 1

Introduction Office 365 • 1

It's Learning • 10, 273

## L

LDAP Attributes • 9

LDAP attributes - Attribute specification • 365, 376

LDAP attributes - Data conversion • 367, 368

LDAP attributes - Data conversion routine • 367, 368

LDAP binding • 6

LDAP Directory Service - Encrypt input • 370

LDAP Directory Service - LDAP Search • 372

LDAP Directory Service - LDAP Search Attributes • 372

LDAP Directory Service - Setup LDAP modification data • 373

LDAP directory services • 196

LDAP filter • 7

LDAP search - Attributes • 363, 365, 374, 378, 381

LDAP search - LDAP binding • 363, 374, 377, 379, 381

LDAP search - LDAP Filter • 363, 368, 374, 378, 379, 381

LDAP search - Options • 378, 381

License code • 364, 382, 385, 386

License matrix • 385

License model • 361, 382, 384, 428

Licensing • 12

Linking Active Directory to other information systems • 6

Linking the auxiliary project to the main project • 13

Linux OpenLDAP • 39

LoadFormProject • 13

Loading LDAP modification data • 20

Log information • 12, 387

Lotus Notes • 20, 202

Lotus Notes Document Item Specification • 389

Lotus Notes example projects • 11, 12, 43, 226, 246, 251

Lotus Notes Item Specification

- General • 390

Lotus Notes Settings dialog • 391

Lotus Notes user guide • 30, 391, 392

## M

Mail • 306

Mail contact • 144

Mail user • 131

Mailbox • 165

Mailbox has never been used. • 10

Manage Active Directory with the UMRA Powershell Agent service • 1, 46, 88, 308, 310, 312, 314, 315, 316, 317, 318

Manage script actions • 331, 339, 387

Manage Services - Adding form fields • 6

Manage Services - Form buttons • 10

Manage Services - Form table • 6

Manage Services - Form, part 1 • 6

Manage Services - Link to project Collect Services • 18

Manage Services - Script • 14

Managing Exchange 2003 with the UMRA Powershell Agent service • 2, 46

Managing LDAP directory services using UMRA • 25, 21, 370, 372, 373

Managing printer queues • 27, 51

Managing printers and printer queues • 194

Managing service projects • 392

Managing UMRA PSM • 83

Managing user account group memberships on Novell eDirectory • 36

Managing Windows computer services • 27, 17

Manual installation of the Powershell Agent service • 11, 9

Mass updating network resources - Mass module • 1, 10

Microsoft Active Directory • 49

Miscellaneous UMRA PSM topics • 83

Move mail files to another server • 39, 43

MS Access database • 26

Multi-value Dependent Action Property script phrase • 35

## N

N@tSchool • 10, 259

Name generation • 83, 5, 6, 13, 39, 284

Name Generation

Default input names • 394, 399

Embedded algorithms • 395

Formatting functions • 395, 397, 399

Iteration • 395

Manage algorithms • 395, 396, 399

Setup algorithm methods • 395, 398

Name Generation Algorithms • 393, 394, 395, 396, 397, 399

Network bar - Count users • 29, 397

Network data • 398

Network table • 5

No help available • 431

non- Active Directory • 38

Novell eDirectory • 13

## O

Office 365 • 3

Office 365 Users • 1

Open UMRA project • 398

Other actions • 187

Out-Of-Office • 98, 168

Output specification • 25

Overview • 79

## P

Password generation • 399

Password Synchronization Manager • 78

Password Synchronisation Manager service settings • 326

Powershell • 308

Powershell Agent connection settings • 3, 4, 45

Powershell Agent service • 19, 1

Powershell Agent service - Edit connection settings • 4, 5

Powershell Agent service session • 10, 13, 50, 51, 53, 321, 322

Powershell Agent service setup - Procedure • 3

Powershell Agent service setup - Requirements • 2, 6

Powershell Agent service wizard - Delete all files • 9

Powershell Agent service wizard - Manage • 4, 5

Powershell Agent service wizard - Specify account • 7

Powershell Agent service wizard - Specify account group • 7

Powershell Agent service wizard - Specify port number • 6

Powershell Agent service wizard - Specify server • 5

Powershell Agent service wizard - Specify service directory • 6

Powershell Agent service wizard - Specify user account group • 8

Powershell Agent service wizard - Update Powershell Agent service • 5, 8

Powershell snap-ins • 1, 12, 18, 36, 46, 88

Prerequisites • 2, 79

Principle of operation • 2

Print jobs project - Form • 3

Print jobs project - Form buttons • 7

Print jobs project - Script • 9

Print jobs project - Table with printer documents • 4

Processing user input • 6, 9, 16

Programmatically creating and evaluating tables • 13, 16

Programming • 301

Project A - Collecting services • 26

Project B - Inserting a form table to display table content in a variable • 28

Project definition • 1

Project description • 1

Project execution • 14, 18

Project extensions • 15

Project principle • 2

Project structure • 1

Properties section • 19

Properties specification • 19

PropertiesSeriesSet section • 23

## Q

QuoteFormat of a Value Dependent Action  
Property script phrase • 36

## R

Reboot options • 325

References • 50, 81

Refresh options • 325

Registry settings • 13

Release notes • 1

ReleaseConnection • 13

Remove a dynamic action • 40

Requirement UMRA Exchange 2007 support • 1

RestoreConnection • 13

ReturnData element specification • 27

## S

SAP - Action

SAP Setup connection • 73

SAP - Connections • 74

SAP - Example projects • 75



- 
- SAP - Registry settings • 76
  - SAP - Requirements • 73
  - SAP - SAP Generic function module • 8, 74
  - SAP - UMRA SAP child process • 74
  - SAP actions • 257
  - Scheduler • 401, 405, 410, 411, 412
  - Script action
    - Add directory service object (LDAP) • 3, 4, 6, 11, 373
    - Delete directory service object (LDAP) • 3, 7, 12, 32, 373, 374
    - Delete multiple variables • 13, 298
    - Load LDAP modification data • 3, 4, 7, 11, 373
    - Modify directory service object (LDAP) • 3, 4, 7, 11, 373, 374
    - Search directory service (LDAP) • 3, 8, 12
    - Setup LDAP session • 3, 7, 11, 12, 32, 373
    - Terminal Services user settings • 54, 58
    - Update database • 282
  - Script Action
    - Add account to local group • 35, 54, 50
    - Add AD permission • 17, 310
    - Add directory service object (LDAP) • 25, 55, 198
    - Check Powershell Agent service session • 10, 274, 321
    - Check session variable • 54, 301
    - Configure Out-Of-Office • 17, 231
    - Configure service • 27, 55, 193
    - Connect mailbox (Exchange 2007) • 123
    - Convert text to date/time • 33, 55, 293
    - Convert to multi-value variable • 33, 55, 294, 295
    - Convert value of variable • 27, 55, 292
    - Copy directory • 55, 174, 182, 298
    - Copy document • 14, 240
    - Count licensed - domain/OU accounts • 20, 188
    - Create (enable) mailbox (Exchange 2007) • 104
    - Create contact (AD) • 32, 54, 11
    - Create Directory • 35, 55, 3, 6, 22, 38, 171, 177, 182, 287, 414, 416, 417
    - Create distribution group (Exchange 2007) • 160
    - Create document • 239
    - Create Exchange Mailbox (2003/2000) • 23, 54, 88, 92, 93, 94, 95, 96, 98
    - Create group (AD) • 22, 32, 54, 52, 78
    - Create local group • 52
    - Create mail contact (Exchange 2007) • 144
    - Create mail user (Exchange 2007) • 131
    - Create object (AD) • 22, 54, 66, 68
    - Create share • 33, 55, 182, 184, 185

Create user (AD) • 22, 33, 35, 54, 3, 31, 32, 38, 49, 50, 51, 88, 171, 284, 285, 286, 294, 336, 394, 395, 399, 400

Create User (no AD) • 33, 35, 54, 3, 38, 45, 49, 171, 284, 285, 294, 394, 395, 399, 400

Create user and mailbox (Exchange 2007) • 101

Delay • 56, 306

Delete attribute value (AD) • 25, 54, 72

Delete directory • 55, 174, 177, 180, 185, 298

Delete directory service object (LDAP) • 25, 55, 199

Delete document • 12, 240

Delete Exchange mailbox (2000/2003) • 54, 90, 92, 94, 95, 98

Delete file(s) • 179

Delete Item • 251

Delete Object (AD) • 27, 54, 67

Delete person • 223, 241

Delete session variable • 54, 301

Delete share • 33, 55, 183, 184, 185

Delete user (AD) • 54, 30, 48, 185

Delete user (no AD) • 54, 31, 48, 185

Delete variable • 56, 297, 340

Dial-in user settings • 32, 54, 64

Disable distribution group (Exchange 2007) • 164

Disable mail contact (Exchange 2007) • 152

Disable mail user (Exchange 2007) • 140

Disable mailbox (Exchange 2007) • 120

Edit distribution group (Exchange 2007) • 164

Edit Exchange mailbox (2000/2003) • 54, 90

Edit mail contact (Exchange 2007) • 147, 151

Edit mail user (Exchange 2007) • 135, 140

Edit mailbox (Exchange2007) • 105, 114

Edit person • 215, 236, 246

Edit share • 13, 27, 55, 183, 185

Edit user (AD) • 33, 54, 19, 43, 45, 90, 336

Edit user (no AD) • 33, 54, 43

Edit user logon • 32, 54, 25, 45, 54

Enable distribution group (Exchange 2007) • 162

Enable mail contact (Exchange 2007) • 146

Enable mail user (Exchange 2007) • 134

Encrypt text • 56, 287, 298

Execute agent script • 11, 43, 255

Execute Command Line • 12, 55, 187

Execute print job command • 27, 55, 195

Execute script • 25, 56, 303

Execute service command • 27, 55, 192, 194

Export Variables • 33, 34, 56, 204, 209, 296, 299, 300

For-Each • 22, 30, 56, 304

Format Variable Value • 55, 288, 395

Generate generic table • 21, 23, 55, 275, 430

Generate name(s) • 33, 55, 284

Generate password • 25, 56, 204, 209, 299

Generate random number • 33, 56, 299

Generate recovery password • 17, 227

Get (nested) group memberships • 17, 318

Get AD permissions • 17, 308

Get attribute (AD) • 54, 68

Get certifier • 202, 203, 208, 221, 222, 225, 336

Get database • 226, 233, 234, 237, 239, 240, 243, 244, 251, 254

Get databases • 234

Get disk space • 17, 319

Get document • 216, 221, 223, 224, 225, 226, 230, 236, 238, 239, 241, 242, 243, 246, 247, 251, 402

Get documents • 235, 237, 244

Get file/directory info • 55, 173, 187

Get item • 241

Get item size • 11, 242

Get mailbox permissions (Exchange 2007) • 126

Get Object (AD) • 33, 54, 68, 70, 71, 73, 76, 81

Get Out-Of-Office info (Exchange 2000/2003) • 13, 98, 100

Get Out-Of-Office info (Exchange 2007) • 10, 168

Get owner • 17, 315

Get PDC (AD) • 17, 321

Get primary group • 30, 55, 87

Get quota • 14, 231

Get session variable • 54, 300

Get terminal services user settings • 27, 54, 63

Get user (AD) • 54, 17, 19, 20, 25, 26, 30, 31, 36, 45, 46, 54, 55, 58, 68, 70, 71, 73, 82, 84, 85, 91, 92, 95, 96, 99, 100, 280, 282, 297

Get user info • 27, 54, 30, 56

Get user table (locked out/disabled/password) • 25, 54, 27, 72, 430

Get variable length • 288

Get views • 235

Go to Label • 56, 302, 306, 389

If-Then- Else • 31, 56, 291, 303

Join table data • 21, 278

List contacts (Exchange 2007) • 158

List distribution groups (Exchange 2007) • 164

List files and/or directories • 25, 185, 430

List mail contacts (Exchange 2007) • 153

List mail users (Exchange 2007) • 142

List mailbox databases (Exchange 2007) • 166

List mailbox statistics (Exchange 2007) • 121, 165

List mailboxes (Exchange 2007) • 129

List printer documents • 27, 55, 194, 195, 196, 430

List services status • 27, 55, 189, 192, 194, 430

List users (Exchange 2007) • 156

Load LDAP modification data • 25, 55, 197, 198, 199

Log Specific Variables • 300

Log Variables • 56, 299, 300

Manage Exchange recipient mail addresses (2003/2000) • 32, 54, 90, 92, 94, 95, 96

Manage mail contact email addresses (Exchange 2007) • 151

Manage mail user email addresses (Exchange 2007) • 139

Manage mailbox email addresses (Exchange 2007) • 113

Manage mailbox permissions (Exchange 2007) • 127

Manage multi-text value variable • 33, 55, 76, 295

Manage table data • 13, 14, 21, 22, 31, 55, 238, 245, 275, 276, 282, 291, 305, 363

Map variable • 56, 32, 50, 301, 302, 303, 332, 350

Merge multi-text variable values • 31, 56, 295

Modify directory service object (LDAP) • 25, 55, 198, 199

Modify Exchange mailbox permissions (2000/2003) • 32, 54, 90, 92, 95, 96, 98

Move - rename (AD) • 54, 20, 25, 27, 31, 35, 37, 38, 46, 48, 54, 56, 84, 87, 298

Move cross-domain (AD) • 27, 54, 37, 86

Move Exchange mailbox • 27, 54, 90, 92, 94, 95, 96, 98

Move mailbox (Exchange 2007) • 124

Move person • 216, 224

Move person (advanced) • 17, 225

No operation • 56, 306

Process all requests • 232

Query Document Items • 244

Recertify person • 222

Recover ID file • 228

Register person • 202, 203, 208, 216, 221, 223, 239

Register person (advanced) • 11, 208

Release Powershell Agent service session • 51, 274, 321, 322

Remove AD permission • 17, 312

Remove distribution group (Exchange 2007) • 164

Remove group member • 31, 54, 52

Remove mail contact (Exchange 2007) • 153

Remove mail user (Exchange 2007) • 141

Remove SID history • 24, 73

- 
- Remove specific group memberships (AD) • 31, 54, 77
  - Remove user - mailbox (Exchange 2007) • 121
  - Remove user group memberships (AD) • 54, 33, 77
  - Rename directory service object (LDAP) • 24, 200
  - Rename file or directory • 31, 55, 177
  - Rename person • 221, 225
  - Search documents • 216, 221, 223, 224, 225, 226, 230, 237, 243
  - Search LDAP • 25, 55, 201
  - Search object (AD) • 54, 17, 82
  - Send HTML mail message • 6, 7, 307
  - Send mail message • 31, 56, 306
  - Set AD permissions (advanced) • 17, 314
  - Set attribute (AD) • 12, 13, 32, 54, 70, 280, 282
  - Set client access attributes (Exchange 2007) • 114
  - Set encrypted variable • 25, 55, 202, 274, 287, 298, 371
  - Set group membership (AD) • 32, 54, 76, 294
  - Set Internet password • 229
  - Set item(s) • 246
  - Set Managed By • 17, 317
  - Set Out-Of-Office info (Exchange 2000/2003) • 13, 100
  - Set Out-Of-Office info (Exchange 2007) • 10, 169
  - Set owner • 17, 316
  - Set primary group (AD) • 31, 54, 55, 54, 88
  - Set primary group (non AD) • 53
  - Set quota • 230
  - Set session variable • 54, 301
  - Set user group memberships (AD) • 35, 54, 31, 34, 50, 51
  - Set Variable • 13, 32, 55, 32, 286, 292, 299, 301, 332
  - Setup LDAP session • 25, 55, 196, 198, 199
  - Setup Powershell Agent service session • 51, 274, 321, 322
  - Setup Security • 55, 178
  - Setup User Global Group Memberships • 54, 34, 49, 332
  - Sign/Unsign document • 245, 248, 391
  - Split Variable • 55, 287
  - Update ACL • 254
  - Update database - Database • 55, 282, 284
  - Update database - Introduction • 27, 55, 282
  - Update database - SQL Statements • 9, 55, 282, 284
  - Update database - Test • 55, 282, 283, 284
  - Update date-time variable • 12, 23, 27, 55, 291
  - Update group memberships (AD) • 15, 74

- 
- Update numeric variable • 12, 28, 31, 55, 289
  - Update profile document • 11, 14, 17, 251
  - Script action 1
    - Check %DocumentID% • 9
  - Script action 2
    - Check %PrinterCommand% • 10
  - Script action 3
    - Go-To printer command • 11
  - Script action 4
    - Execute print job command • 12
  - Script action 5
    - Reset %DocumentID% • 13
  - Script action overview • 3
  - Script action property value • 400
  - Script action property value - Output only • 413
  - Script action property value output • 412, 413
  - Script action property value with yes/no option • 412
  - Script execution • 10, 15, 16
  - Script phrase contents • 37
  - Script section • 28
  - Scripts, actions and properties • 4
  - Search and replace • 413
  - Searching a directory service (LDAP) • 12
  - Searching accounts and resetting passwords in Microsoft Active Directory using LDAP • 49, 63
  - Secure LDAP Active Directory environment • 49
  - Secure LDAP eDirectory environment • 13
  - Secure Linux OpenLDAP environment • 39
  - Security - Access Control Settings • 414
  - Security - Adding accounts and permissions • 415, 416
  - Security - Detailed permissions settings • 415
  - Security - Overview • 177, 414, 415, 416, 417
  - Security - Owner • 417
  - Security and authentication • 34
  - Select domain controller wizard - Specify the target domain • 325
  - Select domain controller wizard - welcome • 325
  - Selecting data using form tables • 3
  - Series of properties • 23
  - Session section • 38
  - Set items • 402
  - SetCellText • 21
  - SetColumnName • 22
  - SetColumnNameEx • 22
  - Setting a user account password on Novell eDirectory • 24
  - Setting up a secure session with Active Directory domain controller • 58

- 
- Setting up a secure session with Linux LDAP Server • 44, 45
  - Setting up an LDAP session • 19
  - Setting up the Exchange 2007 Management Tools on a 32-bit platform • 1, 2, 3, 46, 88
  - Setting up the IIS website • 42
  - Setting up the List services status action • 3
  - Setting up the Set variable action • 3
  - Setting up user account group memberships on Novell eDirectory • 32
  - Setup project security • 4
  - Setup scheduling • 401, 402, 410, 411, 412
  - Setup scheduling - Adding an exception • 402, 405, 410, 412
  - Setup scheduling - Exceptions • 401, 405, 411, 412
  - Setup scheduling - Preview • 401, 402, 405, 410, 411, 412
  - SetVariableBool • 14
  - SetVariableLong • 14
  - SetVariableTable • 14
  - SetVariableText • 15
  - Signature of UMRA dynamic actions • 18, 40
  - Simple script phrase • 30
  - Simple Value Dependent Action Property script phrase • 34
  - Single value output data • 17, 25
  - SOAP Synchronization template project • 100, 263
  - Special table type - Generic table Variable • 5, 13
  - Specify file input data • 363, 417
  - Specify group names • 418
  - Specify input • 418
  - Specify input name • 418
  - Specify new name for UMRA project • 418
  - Specify radio button text info • 418
  - Specify the name of the domain controller • 326
  - Specify variable info • 419
  - Specifying Active Directory LDAP attributes • 58, 59
  - Specifying columns • 13
  - Specifying columns for table type Variable • 15
  - Specifying LDAP attributes and values of directory service item • 44, 45
  - Starting the UMRA Console • 2
  - Step 1
    - Environment setup • 2
    - Obtain a certificate • 55
  - Step 1 - Designing the form layout • 28
  - Step 1 - Importing your input data • 10
  - Step 2
    - Form project - Collect Services • 2
  - Step 2 - Building the form script • 46
  - Step 2 - Building the project script • 13

## Step 2 - Windows Server 2003

Install the certificate in the IIS website • 55

## Step 2 - Windows XP

Install the certificate in the IIS website • 58

## Step 3

Form project - Manage Services • 5

Install the Certification Authority root  
certificate on the UMRA Powershell  
Agent service computer • 67

## Step 3 - Specifying security • 49

## Step 3 - Testing the project script • 22

## Step 4 - Executing the project script • 24

## Summary • 5

## System requirements • 8

## T

Table • 275

Table columns specification • 6

Table value output data • 26

Table variable specification • 6

Task scheduler overview settings • 419

Task scheduler overview window • 419

TeleTOP • 10, 260

Testing the project script • 51

The concept of tables in UMRA • 2

TOPdesk • 10, 257

## U

Umra • 5

UMRA Basics • 24, 3, 83, 16, 19, 25, 27, 31, 33,  
34, 37, 38, 43, 45, 48, 50, 51, 56, 66, 70, 81,  
82, 84, 86, 87, 90, 92, 94, 95, 96, 98, 183, 184,  
185, 280, 281, 331, 332, 338, 347, 351, 364,  
385, 386, 387, 389, 392, 394, 395, 396, 397,  
400, 412, 413, 414, 415, 416, 417, 422, 423,  
426

UMRA COM • 16, 18, 1

UMRA COM - Main functions • 1

UMRA COM in IIS • 33

UMRA COM in VB scripts • 23

UMRA COM object reference • 14, 5, 53

UMRA COM objects • 3

UMRA COM on 64-bit platforms • 18, 34, 48

UMRA command line • 18

UMRA components • 17

UMRA Console • 17

UMRA Console - Command Line Options • 32,  
421

UMRA dynamic action example

Goal • 41

Import the dynamic action • 40, 44

Using the dynamic action • 45

XML file • 41

XML file - ActionProperties section • 43

XML file - general section • 42

XML file - Script section • 43

UMRA dynamic actions • 14

UMRA Forms client • 18



- UMRA Google Module • 8, 70
- UMRA installation • 2
- UMRA LDAP script actions • 3
- UMRA Powershell Agent configuration • 6, 7
- UMRA Project Component - File data • 422
- UMRA Project Component - Form • 423
- UMRA Project Component - Network data • 423
- UMRA Project Component - Preview • 423
- UMRA Project Component - Script • 423
- UMRA project management • 1
- UMRA Project Properties - Description • 423
- UMRA Project Properties - Form Fonts • 424
- UMRA Project Properties - Form options • 28, 33, 424
- UMRA Project Properties - Format • 343, 425
- UMRA Project Properties - General • 425
- UMRA Project Properties - Initial variables • 352, 426
- UMRA Project Properties - Network data • 426
- UMRA Project Properties - Options • 426
- UMRA Project Properties - Security • 426
- UMRA projects • 1
- UMRA Projects • 7, 8, 1
- UMRA projects for managing printer queues • 1
- UMRA PSM Domain Controllers Overview • 322
- UMRA PSM Package installation status • 84, 322
- UMRA PSM Package registry values • 85
- UMRA PSM script variables • 83, 85, 326
- UMRA Reference Guide • 1
- UMRA SAP module • 8, 13, 73, 257
- UMRA scripting • 4
- UMRA Service • 17
- UMRA service - Advanced options • 427
- UMRA service - license • 384, 427
- UMRA Service - service access • 428
- UMRA service deletion - Delete all files • 428
- UMRA service installation - Admin group • 428, 429
- UMRA service installation - Port • 429
- UMRA service installation - Server • 429
- UMRA service installation - Service account • 429
- UMRA service installation - Service directory • 429
- UMRA Sessions • 7, 13, 53, 300, 301, 321
- UMRA setup • 2
- UMRA tables • 28, 30, 7, 30, 195, 348, 351, 353, 355, 363, 364, 377, 378, 381, 382, 417, 430
- UMRA User Guide • 1
- UMRA workspace • 24, 426
- UMRA workspaces • 1
- UMRA XML project and script files • 15, 2
- UmraCheckLicense • 16
- UmraDataTable • 17

UmraFormProject • 16

UmraFormTable • 17

Updating directory service item attributes • 11

Updating group memberships in Microsoft  
Active Directory using LDAP • 49, 78

Upgrading MASS projects from older versions •  
24, 36

Upgrading UMRA dynamic actions • 17, 39

User • 3, 156

User account provisioning - Automation module  
• 6

User mailbox • 101

User Management Resource Administrator  
release notes • 1

Using tables in UMRA - Forms & Delegation -  
Hands-on • 19

Using the built-in name generation algorithms •  
1

Using the Exchange Web Services with UMRA •  
6, 168, 169

Using UMRA COM • 3

## V

Value Dependent Action Property script phrase  
• 32, 33

Value of date-time item • 327

Value of numeric item • 328

Value of text item. • 326

Value of text list item • 327

Variable actions • 275

Variable Conversion of a Value Dependent  
Action Property script phrase • 37

Variable generic table • 363, 373, 429

Variable list • 55, 282, 283, 284, 431

Variable operations • 286

Variables • 6

Viewing the current status of PSM Packages •  
84

Visual Basic script • 26

## W

Welcome to UMRA • 1

Which module should I choose? • 8

Windows computer services • 189

## X

XML file specification • 14