NetIQ

NetIQ® Identity Manager

Driver for Scripting Implementation Guide

July 2023

Contents

About this Book and the Library

This guide describes implementation of the NetIQ® Identity Manager 4.8 driver for Scripting.

The driver synchronizes data from a connected system through a scriptable interface with Identity Manager 4.0, the comprehensive identity management suite that allows organizations to manage the full user life cycle, from initial hire, through ongoing changes, to ultimate retirement of the user relationship.

Other Information in the Library

The library provides the following information resources:

Identity Manager Setup Guide

Provides overview of Identity Manager and its components. This book also provides detailed planning and installation information for Identity Manager.

Designer Administration Guide

Provides information about designing, testing, documenting, and deploying Identity Manager solutions in a highly productive environment.

User Application: Administration Guide

Describes how to administer the Identity Manager User Application.

User Application: User Guide

Describes the user interface of the Identity Manager User Application and how you can use the features it offers, including identity self-service, the Work Dashboard, role and resource management, and compliance management.

User Application: Design Guide

Describes how to use the Designer to create User Application components, including how to work with the Provisioning view, the directory abstraction layer editor, the provisioning request definition editor, the provisioning team editor, and the role catalog.

Identity Reporting Module Guide

Describes the Identity Reporting Module for Identity Manager and how you can use the features it offers, including the Reporting Module user interface and custom report definitions, as well as providing installation instructions.

Analyzer Administration Guide

Describes how to administer Analyzer for Identity Manager.

Identity Manager Common Driver Administration Guide

Provides information about administration tasks that are common to all Identity Manager drivers.

Identity Manager Driver Guides

Provides implementation information about Identity Manager drivers.

About NetIQ Corporation

We are a global, enterprise software company, with a focus on the three persistent challenges in your environment: Change, complexity and risk—and how we can help you control them.

Our Viewpoint

Adapting to change and managing complexity and risk are nothing new

In fact, of all the challenges you face, these are perhaps the most prominent variables that deny you the control you need to securely measure, monitor, and manage your physical, virtual, and cloud computing environments.

Enabling critical business services, better and faster

We believe that providing as much control as possible to IT organizations is the only way to enable timelier and cost effective delivery of services. Persistent pressures like change and complexity will only continue to increase as organizations continue to change and the technologies needed to manage them become inherently more complex.

Our Philosophy

Selling intelligent solutions, not just software

In order to provide reliable control, we first make sure we understand the real-world scenarios in which IT organizations like yours operate — day in and day out. That's the only way we can develop practical, intelligent IT solutions that successfully yield proven, measurable results. And that's so much more rewarding than simply selling software.

Driving your success is our passion

We place your success at the heart of how we do business. From product inception to deployment, we understand that you need IT solutions that work well and integrate seamlessly with your existing investments; you need ongoing support and training post-deployment; and you need someone that is truly easy to work with — for a change. Ultimately, when you succeed, we all succeed.

Our Solutions

- Identity & Access Governance
- Access Management
- Security Management
- Systems & Application Management
- Workload Management
- Service Management

Contacting Sales Support

For questions about products, pricing, and capabilities, contact your local partner. If you cannot contact your partner, contact our Sales Support team.

Worldwide:	www.netiq.com/about_netiq/officelocations.asp
United States and Canada:	1-888-323-6768
Email:	info@netiq.com
Web Site:	www.netiq.com

Contacting Technical Support

For specific product issues, contact our Technical Support team.

Worldwide:	www.netiq.com/support/contactinfo.asp
North and South America:	1-713-418-5555
Europe, Middle East, and Africa:	+353 (0) 91-782 677
Email:	support@netiq.com
Web Site:	www.netiq.com/support

Contacting Documentation Support

Our goal is to provide documentation that meets your needs. If you have suggestions for improvements, click **Add Comment** at the bottom of any page in the HTML versions of the documentation posted at www.netiq.com/documentation. You can also email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

Contacting the Online User Community

Qmunity, the NetIQ online community, is a collaborative network connecting you to your peers and NetIQ experts. By providing more immediate information, useful links to helpful resources, and access to NetIQ experts, Qmunity helps ensure you are mastering the knowledge you need to realize the full potential of IT investments upon which you rely. For more information, visit http://community.netiq.com.

1 Overview

The Identity Manager 4.8 driver for Scripting synchronizes data between the Identity Vault and a connected system using a scripting environment suitable for the target application. Languages such as Shell, Perl* and Microsoft* VBScript* can be used to extend the base set of sample scripts to update and retrieve information from the target application. Traditional driver development is accomplished with Java* and the Policy Builder. With the Scripting environment, driver development can be done in alternate languages, which provide a large set of packages and complex language syntax. In addition, your target application might already provide management tools, commands, or APIs written or easily accessible in Perl, Shell Script or VBScript.

The Scripting driver runs on a large number of Linux and UNIX platforms, including Linux, Solaris*, AIX* and HP-UX*. In addition, a Microsoft Windows* service is available for applications that run on the Windows platform. The driver also uses embedded Remote Loader technology to communicate with the Identity Vault, bidirectionally synchronizing changes between the Identity Vault and the connected system. The embedded Remote Loader component, also called the driver shim, runs as a native process on the connected Linux or UNIX system or a Windows service on a Windows system. There is no requirement to install Java on the connected system. The simplicity of installation, configuration and security provides an excellent environment for the development of new drivers.

Major topics in this section include

- "Driver Architecture" on page 11
- "Configuration Overview" on page 13

Driver Architecture

The Scripting driver synchronizes information between the Identity Vault and an external account management system (the connected system).

The Identity Manager detects relevant changes to identities in the Identity Vault and notifies the Subscriber component of the driver. After customizable policy processing, events are sent to the Subscriber shim of the embedded Remote Loader process on the connected system. The Subscriber shim securely passes the information to customizable scripts that perform the required actions.

A process on the connected system polls the account management system for changes at a configurable interval. If the poll returns identity changes, they are written to the change log.

The Publisher shim of the embedded Remote Loader process submits the changes from the change log to the Metadirectory engine as events. The Metadirectory engine processes these events using customizable policies and posts relevant changes to the Identity Vault.

Topics in this section include

- "Publisher Channel" on page 12
- "Subscriber Channel" on page 13
- "Scriptable Framework" on page 13

* “Schema File” on page 13
* “Include/Exclude File” on page 13
* “Loopback State Files” on page 13

Publisher Channel

The Publisher shim provides identity change information to the Metadirectory engine as XDS event documents. The Metadirectory engine applies policies, takes the appropriate actions, and posts the events to the Identity Vault.

Change Log

The change log stores identity changes in encrypted form. The polling script uses the change log update command to record identity changes it detects. Events are removed from the change log by the Publisher shim at configurable intervals and submitted to the Metadirectory engine for processing. If communication with the Metadirectory engine is temporarily lost, events remain in the change log until communication becomes available again.

Change Log Update Command

The change log update command encrypts and writes events to the change log. Any process with rights to update the change log can use the change log update command. The change log update command takes command line arguments and standard input, and stores events in encrypted form in the change log for subsequent publishing. The polling script calls the change log update command to record identity changes. For information about using the change log update command, see the developer guides in Chapter 5, “Customizing the Scripting Driver,” on page 35.

Polling Script

The polling script periodically scans the local account management system for modifications that have occurred since the last polling interval. If necessary, the polling script updates the change log by calling the change log update command. You can specify the polling interval during installation and by subsequent configuration of the Driver object.

Publisher Shim

The Publisher shim periodically scans the change log for events. Before scanning the change log, the driver calls the polling script to check the local system for changes that might have been made since the previous poll.

When the Publisher shim finds events in the change log, it decrypts, processes, and sends them to the Metadirectory engine in XDS format over a Secure Sockets Layer (SSL) network link.

Subscriber Channel

The Subscriber channel receives XDS command documents from the Metadirectory engine and calls the appropriate script or scripts to handle the command.

The provided scripts must be customized to handle connected system events. For more information see Chapter 5, “Customizing the Scripting Driver,” on page 35.

Scriptable Framework

The interface between the connected system and the driver shim uses customizable scripts. You must extend the scripts that are provided with the driver to support your connected system. Several utility scripts and helper commands are provided with the driver to facilitate communication with the driver shim and the change log. An extensible connected system schema file allows you to add your own objects and attributes to those already supported by the driver.

For more information about the scriptable framework, see Chapter 5, “Customizing the Scripting Driver,” on page 35.

Schema File

The configuration of class and attribute definitions for the connected system is specified using the schema file. You can modify and extend this file to include new objects and attributes. For details about configuring the schema file, see “The Connected System Schema File” on page 38.

Include/Exclude File

The include/exclude file allows local system policy to enforce which objects are included or excluded from provisioning, on both the Publisher channel and the Subscriber channel, independently. For details about using the include/exclude file, see “The Connected System Include/Exclude File” on page 40.

Loopback State Files

The loopback state files are used to provide automatic loopback detection for external applications that do not have mechanisms to perform loopback detection. This loopback detection prevents subscribed events from being published back to the Identity Vault.

Configuration Overview

This section discusses driver configuration details specific to the Scripting driver. For basic configuration information, see “Managing Identity Manager Drivers” in the *Identity Manager Administration Guide*. For detailed information about configuring the Scripting driver, see Chapter 4, “Configuring the Scripting Driver,” on page 25.

Topics in this section include

- “Data Flow” on page 14
- “Policies” on page 14

Data Flow

Filters and policies control the data flow of identities to and from the connected system and the Identity Vault. The Data Flow option, specified during driver import, determines how these filters and policies behave.

- **Bidirectional:** Sets classes and attributes to be synchronized on both the Subscriber and Publisher channels.
- **Application to Identity Vault:** Sets classes and attributes to be synchronized on the Publisher channel only.
- **Identity Vault to Application:** Sets classes and attributes to be synchronized on the Subscriber channel only.

Policies

The Metadirectory engine uses policies to control the flow of information into and out of the Identity Vault. Policies can be customized to support desired operations. The following table describes the policy functions for the Scripting driver in the default configuration:

Table 1-1 *Default Linux and UNIX Driver Policy Functions*

Policy	Description
Mapping	Maps the Identity Vault objects and selected attributes to connected system objects and attributes.
Publisher Event	Processes Publisher-side operations.
Publisher Matching	Restricts privileged accounts and defines matching criteria for placement in the Identity Vault.
Publisher Create	Defines creation rules for provisioning into the Identity Vault.
Publisher Placement	Defines where new objects are placed in the Identity Vault.
Publisher Command	Defines password publishing policies.
Subscriber Matching	Defines rules for matching identities in the connected system.
Subscriber Create	Defines required creation criteria.
Subscriber Command	Transforms attributes and defines password subscribing policies.
Subscriber Output	Sends e-mail notifications for password failures and converts information formats from the Identity Vault to the connected system.
Subscriber Event	Restricts events to a specified container.

2 Planning for the Scripting Driver

The planning process for the NetIQ® Identity Manager Driver for Scripting begins by determining whether you develop your own scripts and policies for use with your external account management system or you obtain them from a third party.

If you are customizing the scripts, the process continues by installing the Scripting driver to a development system. The first decision to make is whether you run the driver shim on Windows or a Linux/UNIX system. (The driver engine component hosted by NetIQ Identity Manager can run on any operating system that supports Identity Manager.) The driver shim should be installed on a system that hosts, or is remotely connected to, the external account management system. The operating system of this system is the operating system of the driver shim.

Your operating system choice determines what scripting language to use. On Linux and UNIX, pre-written libraries are included for Bourne shell, Perl and Python*. On Windows, libraries are included for Microsoft VBScript and PowerShell*. Also, with additional development work, you can port these libraries to another scripting language.

Continue the development process by reading Chapter 5, "Customizing the Scripting Driver," on page 35 and following its instructions.

If you have obtained custom scripts from a third party, you should follow their instructions for installing the Scripting driver and their custom components.You should install and test the driver on a test system first and then on production systems.

This section reviews some of the issues to consider before you install the Scripting driver. Major topics include

- "Prerequisites for Linux and UNIX Scripting" on page 15
- "Prerequisites for Windows Scripting" on page 16
- "Establishing a Security-Equivalent User" on page 17

For general information about planning for identity management, see the *Identity Manager 3.6.1 Installation Guide*.

Prerequisites for Linux and UNIX Scripting

Topics in this section include

- "Identity Vault Server Requirements" on page 15
- "Other Software" on page 16

Identity Vault Server Requirements

- NetIQ Identity Manager 4.8.

Other Software

- NetIQ Identity Console
- NetIQ Designer

Prerequisites for Windows Scripting

- “Identity Vault Server” on page 16
- “Windows PowerShell” on page 16
- “Microsoft .NET Framework” on page 17
- “Other Software” on page 17

Identity Vault Server

The following software is required on the Identity Vault Server:

- NetIQ Identity Manager 4.8 or higher
- NetIQ eDirectory (level supported by Identity Manager and higher)

Windows PowerShell

Windows PowerShell is installed by default for Windows Server 2008 R2 and later for servers and Windows 7 and later for workstations. If you are using an older version of Windows, see Microsoft’s website (http://www.microsoft.com) for information on installing PowerShell.

Also be aware of the following:

- Windows PowerShell versions up to 5.x are supported; the latest 5.x version of PowerShell supported by your Windows edition is recommended. Note that the cross-platform versions of PowerShell, i.e. 6.x and higher, are not supported.
- You must use the x86 or x64 version of PowerShell that corresponds with the version of the Scripting Driver you intend to use.
- Before using the Scripting Driver, you must change PowerShell’s default script execution policy as follows:
    1. Right-click **Windows Powershell** (in the **Windows PowerShell** folder) on the Windows Start Menu and select “Run as Administrator”.

       NOTE: You must have administrative privileges to run this command.

    2. Enter the following command:

       ```
       Set-ExecutionPolicy RemoteSigned
       ```

       The setting is saved automatically.

       NOTE: You must have administration privileges to run this command.

    3. Close PowerShell.

Microsoft .NET Framework

- If you are running the Script Service for Windows PowerShell ("Running the Script Service for PowerShell (default mode)" on page 21), the Microsoft .NET Framework is required.
- By default, the Microsoft .NET Framework 4.0 or higher is required to use the Script Service--the version installed by default on your Windows system is recommended. Additionally, you will need to install the .NET Framework 3.5 Feature--the steps to do so are detailed in the installation instructions ("Using the Script Service" on page 21). Alternatively, if you need to support PowerShell version 2.0, you can use the legacy Script Service by following these instructions:
    - 1. Stop the Script Service if it is running.
    - 2. Backup the following files in the `WSDriver\bin` directory:
        - ScriptService.exe
        - ScriptClient.exe
        - SSLogger.dll
        - SSConfFile.dll
    - 3. From the source media, extract the contents of the `Win\win_all_scriptservice_ps20.zip` file to the `WSDriver\bin` directory. This will overwrite the files above.
    - 4. Start the Script Service.
- **NOTE: the highest version of PowerShell that the legacy Script Service will run is version 2.0, even if later versions are installed on the system.**

Other Software

- If you use the Script Service for Windows PowerShell, the Microsoft Web Services Enhancement module must be installed from the installation media--see the installation instructions for more information.
- The latest version of NetIQ Identity Console can be installed on the Identity Vault Server or a separate system.
- NetIQ Designer (optional).

Establishing a Security-Equivalent User

The driver must run with security equivalent to a user with sufficient rights. You can set the driver equivalent to ADMIN or a similar user. For stronger security, you can define a user with only the minimal rights necessary for the operations you want the driver to perform.

The driver user must be a trustee of the containers where synchronized identities reside, with the rights shown in Table 2-1. Inheritance must be set for [Entry Rights] and [All Attribute Rights].

Table 2-1 Base Container Rights Required by the Driver Security-Equivalent User

Operation	[Entry Rights]	[All Attribute Rights]
Subscriber notification of account changes (recommended minimum)	Browse	Compare and Read
Creating objects in the Identity Vault without group synchronization	Browse and Create	Compare and Read
Creating objects in the Identity Vault with group synchronization	Browse and Create	Compare, Read, and Write
Modifying objects in the Identity Vault	Browse	Compare, Read, and Write
Renaming objects in the Identity Vault	Browse and Rename	Compare and Read
Deleting objects from the Identity Vault	Browse and Erase	Compare, Read, and Write
Retrieving passwords from the Identity Vault	Browse and Supervisor	Compare and Read
Updating passwords in the Identity Vault	Browse and Supervisor	Compare, Read, and Write

If you do not set Supervisor for [Entry Rights], the driver cannot set passwords. If you do not want to set passwords, set the Subscribe setting for the User class nspmDistributionPassword attribute to Ignore in the filter to avoid superfluous error messages. For details about accessing and editing the filter, see the documentation about policies on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/). For complete information about rights, see the *NetIQ eDirectory Administration Guide*.

3 Installing the Scripting Driver

This section provides the information you need to install the Identity Manager 4.8 driver for Scripting.

Major topics include

Installing the Linux and UNIX Scripting Driver Shim

Topics in this section include

Installing the Linux and UNIX Scripting Driver Shim

1. Log in to the target application server as `root`.
2. Obtain the `<os>_scriptdriver_install.bin` file from your installation media and execute this self-extracting file on your Linux or UNIX system.
3. Specify a language choice.
4. Read and accept the license agreement.
5. After the package is installed onto your system, you are prompted to enter Driver and Remote Loader passwords. These passwords are used to verify that an authorized driver shim is communicating with the Identity Manager engine. Follow the prompts:

   5a Enter and confirm the Remote Loader password.

   5b Enter and confirm the driver password.
6. Next, you are prompted to retrieve an SSL certificate. NetIQ® eDirectory™ must be running to retrieve the certificate. The certificate allows SSL encryption between the Identity Manager engine and the driver shim. Enabling SSL is optional but is recommended for better security. To retrieve the certificate, follow the prompts:

   6a Specify the DNS name or IP address of your eDirectory server.

   6b Specify the LDAP secure port, default 636.

   6c Enter Y to accept the certificate.
7. You are prompted for a Scripting language to be used on this system. Enter `Perl` for the sample Perl scripts to be installed or enter `Shell` for the sample Bourne Shell scripts to be installed.

8 If you select Perl, you are optionally asked to install the Perl IDMLib perl module into the Perl system path to be accessible by the sample Perl scripts. Enter `Yes` or `No` to install this library.

9 The installation of the driver shim is finished, with the option of starting the Driver Shim Service. Proceed to the next section to complete the installation of the driver.

Running the Driver Shim

Start the driver engine component in NetIQ Identity Console.

The driver shim is a UNIX daemon process. Use the UNIX startup script `usdrvd` to start and stop the NetIQ Identity Manager Linux and UNIX Script Driver (see Chapter 6, "Using the Scripting Driver," on page 121.)

Installing the Windows Scripting Driver

Topics in this section include

- "Installing the Driver Shim" on page 20
- "Running the Driver" on page 21
- "Running the Script Service for PowerShell (default mode)" on page 21
- "Running Multiple Instances of the Driver (optional)" on page 22

Installing the Driver Shim

1 Obtain one of the following files from your installation media:
   - `win_x86_64_scriptdriver_install.exe` (64-bit)
   - `win_x86_scriptdriver_install.exe` (32-bit)

   Run this file on your Windows system.

2 Click **Next** to continue the installation.

3 Accept the default installation folder or specify your own. Click **Next** to continue.

4 Review your settings and click **Next** to continue.

5 After the driver files are copied, you are prompted to retrieve an SSL certificate. NetIQ eDirectory must be running to retrieve the certificate. The certificate allows SSL encryption between the Identity Manager engine and the driver shim. Enabling SSL is optional but is recommended for better security. To retrieve the certificate, click **Yes** and follow the prompts in the console window:

   5a Specify the DNS name or IP address of your eDirectory server.

   5b Specify the LDAP secure port, default 636.

   5c Enter Y to accept the certificate.

6 You are prompted to enter Driver and Remote Loader passwords. These passwords are used to verify that an authorized driver shim is communicating with the Identity Manager engine. Although you don't need to enter the passwords immediately, they must be set at some point before running the driver. Click **Yes** to the prompt and follow the prompts in the console window:

  6a Enter and confirm the Remote Loader password.

  6b Enter and confirm the Driver password.

7 The installation of the driver shim is finished, with the option of starting the Driver Shim Service. Proceed to the next section to complete the installation of the driver.

Running the Driver

Start the driver engine component in NetIQ Identity Console.

The driver shim is a Windows service. Use the Windows Services application to start and stop the NetIQ Identity Manager Windows Script Driver service (see Chapter 6, "Using the Scripting Driver," on page 121).

Running the Script Service for PowerShell (default mode)

The Script Service preloads Windows PowerShell and keeps it in memory to provide faster performance. Requests to the Script Service are securely submitted by a small program called the Script Client.

Using the Script Service

To install and use the Script Service:

1 If necessary, add the .NET Framework 3.5 Feature using the Add Roles and Features operation in the Server Manager application.

2 Run `Win\Microsoft WSE 3.0 Runtime.msi` to install the Web Service Enhancements module.

3 It may be necessary to open TCP port 8081 in your firewall--this port is used only for communication between the Script Client and Script Service executables. The port can be customized in scriptservice.conf, as explained below.

4 In the Services application, set **NetIQ IDM Windows Script Driver - Script Service** to start automatically.

Configuring the Script Service

To configure the Script Service

1 Create a file named `scriptservice.conf` in the `WSDriver\conf` directory.

2 Open the file and add the desired configuration lines, using the following keywords:

Keyword	Description	Syntax
`-address`	Change the default address and port for Script Service. **Default**: localhost:8081	-address <DNS name or IP address>[:port]
`-nosecurity`	Do not enforce security. This command is required if the **Use Windows EFS** driver parameter is disabled.	`-nosecurity`
`-command`	Execute a script command on startup. May be specified multiple times.	`-command <command>`

Using PowerShell Directly

If you no longer wish to use the Script Service, follow these steps:

1 Open the Driver Configuration in Identity Console. In the Driver Parameters, change `Script Command` to `powershell.exe`.

2 Either stop or disable the Script Service.

3 Restart the driver.

Running Multiple Instances of the Driver (optional)

Running multiple instances of the Scripting Driver on the same system may be desirable, but require some additional steps taken. The instructions in this section assume that you have already installed the Scripting Driver on the system.

Adding a Windows Scripting Driver Instance

To add an instance:

1 Copy existing files:

After stopping your original driver, create a new directory, and copy all original driver files to the new directory, using the same directory structure.

For example, copy files and directories from `C:\Program Files\Novell\WSDriver` to `C:\Program Files\Novell\WSDriver2`.

2 Edit `wsdrv.conf`:

2a Open the file `conf\wsdrv.conf` in your new directory structure. Replace all file paths with the path to the new instance directory.

For example, paths might appear for the `-path`, `-tracefile` and `-connection` options.

2b Change the port numbers (connection and HTTP) to be different from the original driver's port numbers.

For example, if the original driver uses default ports 8090 and 8091, the new instance could use 9090 and 9091. Note that these ports need to be opened in a firewall.

3 Create a new service:

Using the Command Prompt, run `wsdriver.exe` from the new instance directory with the following options:

```
wsdriver -installService -instance {number} -path {path}
```

The instance number could be 2 for the second instance, 3 for a third, and so on. The path should be the path to the new instance directory, using quote marks. Here's an example:

```
wsdriver -installService -instance 2 -path "C:\Program
   Files\Novell\WSDriver2"
```

This command will create a service named `Novell IDM Windows Script Driver - 2`.

NOTE: If you would like to run a driver instance directly (not as a service), use the `-instance` option:

```
wsdriver -instance 2
```

This option is not needed for the original instance.

---

4 Create a new driver object:

4a Using Identity Console or Designer, create a new Driver Object to connect to the new instance. Alternatively, export the original driver's configuration and import it as a new driver.

4b After creating the Driver, open its configuration. Change the port number in `Remote Loader Connection Parameters` to the new instance's connection port.

5 Start the services and drivers:

Start the server for the original and new instances, and start the Drivers in Identity Console or Designer. The instances will run independently.

Removing a Windows Scripting Driver Instance

To remove an instance:

1 Stop the Service.

2 Uninstall the Service.

From the Command Prompt, run:

```
wsdriver -removeService -instance {number}
```

For example:

```
wsdriver -removeService -instance 2
```

3 Delete the files.

Delete the new directory, and all sub-directories, created for the instance.

NOTE: To remove the original instance, use the uninstall feature.

Adding a Linux or Unix Scripting Driver Instance

To add an instance:

1 Copy existing files:

After stopping your original driver, copy the file structure from `/opt/novell/usdrv` to your alternate instance location. For example:

```
cp -r /opt/novell/usdrv /opt/novell/usdrv-instance2
```

2 Create a new, separate configuration file from `/etc/usdrv.conf`:

2a Open the new configuration file, `/etc/usdrv-instance2.conf`, and specify the alternate path location for your new instance.

For example, paths might appear for the `-path`, `-tracefile` and `-connection` options.

2b Change the port numbers (connection and HTTP) to be different from the original driver's port numbers.

For example, if the original driver uses default ports 8090 and 8091, the new instance could use 9090 and 9091. Note that these ports need to be opened in a firewall.

3 Create a new, separate, startup script from the original `/etc/init.d/usdrvd` and change the new startup script to specify the alternate paths as well.

4 Create a new driver object:

4a Using Identity Console or Designer, create a new Driver Object to connect to the new instance. Alternatively, export the original driver's configuration and import it as a new driver.

4b After creating the Driver, open its configuration. Change the port number in `Remote Loader Connection Parameters` to the new instance's connection port.

5 Start the services and drivers:

Start the server for the original and new instances, and start the Drivers in Identity Console or Designer. The instances will run independently.

4 Configuring the Scripting Driver

After you have installed the Identity Manager 4.8 driver for Scripting, use the information in this section for configuration. Major topics include

* "Driver Parameters and Global Configuration Values" on page 25
* "The Driver Shim Configuration File" on page 32

Driver Parameters and Global Configuration Values

You can control the operation of the Scripting driver by modifying the properties described in the following sections. Topics in this section include

* "Properties That Can Be Set Only During Driver Import" on page 25
* "Driver Configuration Page" on page 27
* "Global Configuration Values Page" on page 29

IMPORTANT: Changing these values requires a restart of the driver.

To edit the properties shown on the Driver Configuration page and the Global Configuration Values page:

1. In Identity Console, select **Identity Manager Overview** from the Identity Manager task list on the left side of the window.
2. Navigate to your driver set by searching the tree or by entering its name.
3. Click the driver to open its overview.
4. Click the driver icon.
5. Select **Driver Configuration** or **Global Config Values** as appropriate.
6. Edit the property values as desired, then click **OK**.

Properties That Can Be Set Only During Driver Import

Properties that you can set only during driver import are used to generate policies and other configuration details.

To change import-only properties, you must re-import the `Scripting.xml` driver configuration file over the existing driver.

Table 4-1 Driver Import-Only Parameters

Property Name	Values or Format
Data Flow	Bidirectional
	Application to Identity Vault
	Identity Vault to Application
Enable Entitlements	Yes
	No
Use SSL	No

Data Flow

- **Bidirectional:** Identities are synchronized from both the Identity Vault and the connected system (application). After all pending events are processed, the Identity Vault and connected system mirror each other.
- **Application to Identity Vault:** Identities are synchronized from the connected system (application) to the Identity Vault, but not vice versa. For example, an identity created in the Identity Vault is not created on the connected system unless explicitly migrated.
- **Identity Vault to Application:** Identities are synchronized from the Identity Vault to the connected system (application), but not vice versa. For example, changes made to a connected system's identity are not synchronized to the Identity Vault.

Enable Entitlements

Specifies whether the driver uses either Approval Flow or Role-Based Entitlements with the Entitlements Service driver.

Enable entitlements for the driver only if you plan to use the User Application or Role-Based Entitlements with the driver.

You can use Role-Based Entitlements to integrate the Scripting driver with the Identity Manager User Application.

Use SSL

Specifies whether the driver uses Secure Sockets Layer (SSL) to encrypt the connection between the Identity Vault and the application.

We strongly recommend that you use SSL. If you do not use SSL, identity data (including passwords) is sent across the network in clear text.

Driver Configuration Page

Table 4-2 Driver Configuration Page

Property Name	Values or Format
Driver Module	**Connect to Remote Loader** must be selected.
Driver Object Password	Text Value
Authentication ID	Not used by the Scripting driver.
Authentication Context	Not used by the Scripting driver.
Remote Loader Connection	Parameters Host name or IP address and port number of the driver shim on the connected system, and the RDN of the object with server certificate.
Driver Cache Limit	The recommended value is 0 (zero).
Application Password	Not used by the Scripting driver.
Remote Loader Password	Text Value
Startup Option	Auto start
	Manual
Automatic Loopback Detection	Yes
	No
Script Command	Text Value
Script Trace File	Filename
Subscriber Script	Filename
Polling Script	Filename
Heartbeat Script	Filename
Polling Interval	Number of seconds
Heartbeat Interval	Number of seconds

Driver Object Password

The Driver object password is used by the driver shim (embedded Remote Loader) to authenticate itself to the Metadirectory engine. This must be the same password that is specified as the Driver object password on the connected system driver shim.

Remote Loader Connection Parameters

Table 4-3 Remote Loader Connection Parameters

Parameter	Description
host=*hostName*	Connected system host name or IP address.
port=*portNumber*	Connected system TCP port number. The default is 8090.
kmo=*objectRDN*	The RDN of the object with the server certificate signed by the tree’s certificate authority. Enclose the RDN in double quotes (") if the name contains spaces.

The following is an example Remote Loader connection parameter string:

```
hostname=192.168.17.41 port=8090 kmo="SSL CertificateDNS"
```

Remote Loader Password

The Remote Loader password is used to control access to the driver shim (embedded Remote Loader). This must be the same password that is specified as the Remote Loader password on the connected system driver shim.

Automatic Loopback Detection

Specifies whether the driver shim discards events that would cause loopback conditions. This function supplements the loopback detection provided by the Metadirectory engine.

Script Command

Specifies the command line the driver uses when executing scripts. The driver provides default values for Shell scripts, Perl, Python, VBScript and PowerShell. Normally this value does not need to be changed.

Script Trace File

Specifies a file to which script trace output will be written. The path is relative to the Scripting driver installation directory.

Subscriber Script

Specifies the script file that the driver runs for Subscriber events. The driver provides default values for Shell scripts, Python, Perl, VBScript and PowerShell, so this value does not normally need to be changed.

Polling Script

Specifies the script file that the Publisher shim runs to poll for external events. The driver provides default values for Shell scripts, Perl, Python, VBScript and PowerShell, so this value does not normally need to be changed.

Heartbeat Script

Specifies the script file that the Publisher shim runs to check the external account management system's status. The driver provides default values for Shell scripts, Perl, Python, VBScript and PowerShell, so this value does not normally need to be changed.

Polling Interval

Specifies the number of seconds that the Publisher shim waits after running the polling script and sending events from the change log to the Metadirectory engine. The default interval is 60 seconds, and the minimum interval is 1 second.

Heartbeat Interval

Specifies how often, in seconds, the driver shim contacts the Metadirectory engine to verify connectivity. Specify 0 to disable the heartbeat.

Global Configuration Values Page

Table 4-4 *Global Configuration Values*

Property Name	Values or Format
Connected System or Driver Name	Text Value
Auto Associate Objects	Yes
	No
Application Supports Query	Yes
	No
Auto Resolve Association Reference	Yes
	No
Auto Associate Subscriber Value	Text Value
Auto Associated Publisher Value	Text Value
Strip Old Values	Text Value
The Scripting Connected System Accepts Passwords from the Identity Vault	True
	False

Property Name	Values or Format
The Identity Vault Accepts Passwords from the Scripting Connected System	True False
The Identity Vault Accepts Administrative Password Resets from the Scripting Connected System	True False
Publish Passwords to NDS Password	True False
Publish Passwords to Distribution Password	True False
Require Password Policy Validation before Publishing Passwords	True False
Reset User’s External System Password to the Identity Manager Password on Failure	True False
Notify the User of Password Synchronization Failure via E-Mail	True False
User Base Container	Identity Vault Container object
Group Base Container	Identity Vault Container object

To view and edit Password Management GCVs, select **Show** for **Show Password Management Policy**.

To view and edit User and Group Placement GCVs, select **Show** for **Show User and Group Placements**.

Connected System or Driver Name

Specifies the name of the driver. This value is used by the e-mail notification templates.

Auto Associate Objects

Automatically assign associations to objects. Use this option to automatically generate associations on both the Subscriber and Publisher channels.

Application Supports Query

Does your application support the ability to query information? If the driver cannot query data from your target application, objects cannot be matched and information cannot be merged. If you are only interested in event notification to your target scripts and not data synchronization, you can select **No**.

Auto Resolve Association References

Automatically resolve association references from one object to another.

Auto Associate Subscriber Value

When generating associations, select which value should be used for the object’s association. If you choose **SourceName**, the object's naming attribute will be used as the association. This value might be easier to read and use by a script. If you choose **GUID**, the object's Global Unique Identifier will be used. This value is guaranteed to be unique, even when it's renamed or moved.

Auto Associate Publisher Value

When generating associations, select which value should be used for the object’s association. If you choose **SourceName**, the object’s naming attribute will be used as the association. This value might be easier to read and use by a script. If you choose **GUID**, the object's Global Unique Identifier will be used. This value is guaranteed to be unique, even when it is renamed or moved.

Strip Old Values

By default, the engine will remove attribute values whose timestamps are older than that of their current state. Choose **Strip** to keep the default and strip these values when events are processed on the subscriber channel. Choose **Keep** to allow the attribute values to be processed by the subscriber channel. This is useful if your application requires that an action be taken on every event.

The Scripting Connected System Accepts Passwords from the Identity Vault

Specifies whether the driver allows passwords to flow from the Identity Vault to the connected system.

The Identity Vault Accepts Passwords from the Scripting Connected System

Specifies whether the driver allows passwords to flow from the connected system to the Identity Vault.

The Identity Vault Accepts Administrative Password Resets from the Scripting Connected System

Specifies whether the driver allows passwords to be reset from the connected system in the Identity Vault.

Publish Passwords to NDS Password

Specifies whether the driver uses passwords from the connected system to set nonreversible NDS® passwords in the Identity Vault.

Publish Passwords to Distribution Password

Specifies whether the driver uses passwords from the connected system to set NMAS™ Distribution passwords, which are used for Identity Manager password synchronization.

Require Password Policy Validation before Publishing Passwords

Specifies whether the driver applies NMAS password policies to published passwords. If so, a password is not written to the Identity Vault if it does not conform.

Reset User's External System Password to the Identity Manager Password on Failure

Specifies whether, on a publish Distribution Password failure, the driver attempts to reset the password on the connected system using the Distribution Password from the Identity Vault.

Notify the User of Password Synchronization Failure via E-Mail

Specifies whether the driver sends an e-mail to a user if the password cannot be synchronized.

User Base Container

Specifies the base container object in the Identity Vault for user synchronization. This container is used in the Subscriber channel Event Transformation policy to limit the Identity Vault objects being synchronized. This container is used in the Publisher channel Placement policy as the destination for adding objects to the Identity Vault. Use a value similar to the following:

```
users.myorg
```

Group Base Container

Specifies the base container object in the Identity Vault for group synchronization. This container is used in the Subscriber channel Event Transformation policy to limit the Identity Vault objects being synchronized. This container is used in the Publisher channel Placement policy as the destination when adding objects to the Identity Vault. Use a value similar to the following:

```
groups.myorg
```

The Driver Shim Configuration File

The driver shim configuration file controls operation of the driver shim. The location and name of the file is dependent on the operating system:

- Windows: `wsdrv.conf` in the `conf` directory in your installation directory.
- Linux or UNIX: `/etc/usdrv.conf`

A default configuration file is created at installation time.

You can specify the configuration options listed in Table 4-5, one per line. You can also specify these options on the driver shim command line. For details about driver shim command line options, see "Driver Shim Command Line Options" on page 174.

Table 4-5 *Driver Shim Configuration File Statements*

Option (Short and Long Forms)	Description

-conn connString
 	
-connection connString
	A string with connection options. Enclose the string in double quotes ("). If you specify more than one option, separate the options with spaces.
	
	port=driverShimPort
	
	ca=Certificate Authority Key File

-hp httpPort -httpport httpPort	Specifies the HTTP services port number. The default HTTP services port number is 8091. You can connect to this port to view log files. For details, see "Driver Status and Diagnostic Files" on page 129.
-path *driverPath*	Specifies the path for driver files. The default path is `/usr/local/nxdrv`.
-t *traceLevel* -trace *traceLevel*	Sets the level of debug tracing. 0 is no tracing, and 10 is all tracing. For details, see "Driver Status and Diagnostic Files" on page 129. The output file location is specified by the tracefile option.
-tf *fileName* -tracefile *fileName*	Sets the trace file location. Windows default file: `C:\Progra~1\Novell\WSDriver\logs\trace.log` Linux/UNIX default file: `/opt/novell/usdrv/logs/trace.log`.

Option (Short and Long Forms)	Description
-tfm *size* -tracefilemax *size*	Specifies the limit to the size of the trace file for this instance. Specify the value in kilobytes, megabytes, or gigabytes, using the abbreviation for the byte type. The minimum value is 100K. For example: ◆ `-tracefilemax 1000K` ◆ `-tracefilemax 100M` ◆ `-tracefilemax 10G` **NOTE:** ◆ When you add this option to the configuration file, the application uses the specified name for the tracefile and includes up to 9 “roll-over” files. Each file size is 1/10th of the total size specified. The roll-over files are named using the base of the main trace filename plus `_n`, where `n` is 1 through 9. ◆ If the trace file data is larger than the specified maximum when the Driver Shim is started, the trace file data remains larger than the specified maximum until roll-over is completed through all 10 files.

Example Configuration File

```
-tracefile /opt/novell/usdrv/logs/trace.log
-trace 0
-tracefilemax 100M
-connection "ca=/opt/novell/usdrv/keys/ca.pem port=8090"
-httpport 8091
-path /opt/novell/usdrv/
```

5 Customizing the Scripting Driver

This section describes how the scripts can be written to access and manage your target system. Scripting sets for both Linux and UNIX and for Windows provide libraries, accessible by the scripts, to retrieve event and driver data from the driver shim and to return information to the engine for processing.

The Identity Manager engine has some simple requirements to successfully process events. The provided library supplements your scripts to provide the tools necessary for this successful interaction.

Major topics in this section include

Scripting Driver Data Definition

Topics in this section include

Defining Data Classes and Attributes

The first task is to examine your external application's identity data and to determine what data is relevant and how it should be managed. Below is a series of questions to ask in this process. The list is not all-inclusive; there might be other questions you need to consider.

- Are identities stored in a hierarchical or flat format? In a hierarchical system, objects known as containers can contain identities and perhaps other containers, resulting in a tree-like structure. A flat system contains all identities at a single level.
- What types of identities exist in the external application? For example, NetIQ® eDirectory™ contains Users, Groups and Dynamic Groups, to name a few. Identity types, like Groups, are often aggregates of other identity types.
- What uniquely distinguishes (or names) each type of identity? The name is usually an attribute, known as the naming attribute. Systems often use a human-readable name and a unique serial number. You should determine which is suitable for your needs.
- What relationships exist among the identity types? Suppose Groups can contain Users. Can Groups contain other Groups? Are these relationships one-to-one or one-to-many?
- Is there a way for an identity to link to or represent another identity? For example, eDirectory provides Alias objects, which link to other objects. Your driver might need to handle these types of objects.
- What attributes describe each relevant type of object? Which of these attributes is relevant? What is the data type (string, number, etc.) for each attribute? Can the attribute contain multiple values? Are there restrictions on the attribute's values, such as being read-only, or are they restricted to a certain range of values?
- How will this data be synchronized between the Identity Vault and the application? Some attributes will be synchronized one-way (eDirectory-to-application or vice versa), and others will be synchronized bi-directionally.
- Does your application need password synchronization? Will password synchronization be one-way or bi-directional? Restrictions might also apply to other sensitive data. You need to study the APIs for your application to determine how this should be done.
- Are there identity types or specific identities that should be excluded from synchronization? Administrative users are often excluded from synchronization to avoid security issues.
- Does data need to be transformed when it is synchronized? For example, eDirectory stores a person's first name and last name as separate attribute, but an external application might store a name as one attribute. Such transformations can be done in policies or in the scripts.

Through this process, you should make a list of the application's identity types (also known as classes), their attributes, and each attribute's data type and special properties. This list can then be specified in the driver's `schema.def` file:

```
SCHEMA
   CLASS User
     ATTRIBUTE loginName NAMING REQUIRED
     ATTRIBUTE firstName
     ATTRIBUTE lastName
         (etc.)
```

The format of this `schema.def` file is the same regardless of operating system and scripting language. See "The Connected System Schema File" on page 38 for more information.

Associating Identity Vault and Application Classes and Attributes

The next step is to examine the Identity Vault's classes and attributes and determine which best correspond to the external application's identities. If the Identity Vault does not provide a suitable class or attribute, you can define your own by modifying the Identity Vault's schema. For more information, see the *NetIQ eDirectory Administration Guide*.

The driver's driver filter allows the Metadirectory engine to determine which attributes and classes are relevant to the driver. For the Subscriber channel, the engine notifies the driver of changes for only those classes and attributes that are set to Synchronize in the filter. When the driver receives application identity changes on the Publisher channel, Synchronize must be set on classes and attributes for those items to be changed in eDirectory. The easiest way to define a driver filter is to create a new driver with the default XML configuration file provided with the Scripting driver (`Scripting.xml`). In NetIQ Identity Console, edit the Driver Filter to include relevant classes and attributes. Then, export the driver's configuration to an XML file for later use. See "Managing Additional Attributes" on page 45 for more information.

Defining an Association Rule

Each identity needs an association that uniquely identifies that identity for both eDirectory and the external application. The association must be based on information shared between both the Identity Vault and the application. The association is usually based on one or more attribute values. Below are some ideas for forming an association:

- If a naming attribute is unique across all classes, you could use that attribute value.
- If a naming attribute is unique for a specific class, concatenate the attribute and class name to form the association. For example, an identity named "Bob" with the class "User" could have association "BobUser".
- In a hierarchical system (like eDirectory), you could use a name with its complete hierarchical path, assuming that it is unique. For example, an identity with a hierarchical path "Bob.Users.ACME" could use that path as its association.
- A system can provide a serial number for each identity. This unique number can be used for an association.

Defining Excluded Identities

You might have a list of identities that you want to exclude from synchronization. Also, if an identity is synchronized with sensitive information, you might want to reject that identity. The `include-exclude.conf` file allows such specifications:

```
EXCLUDE
   adminUser
   CLASS secureUser
```

The format of this file is the same regardless of operating system and scripting language. See "The Connected System Include/Exclude File" on page 40 for more information.

Defining Relevant Events

When data changes in an identity management system, an event is said to have occurred. In preparation for the next step, evaluate which event types are relevant for your application:

- **Add:** An identity is created. All required attributes must be created. Security parameters, such as a password, should be defined for the identity to ensure that security isn't compromised.
- **Modify:** One or more attributes of an existing identity are changed. This might affect identities that have a relationship to the changed identity.
- **Modify-password:** An identity's password has changed. This event type can be considered a subtype of Modify, but because it often requires special handling, it is treated as a separate event type.
- **Delete:** An identity is destroyed. This might mean permanent deletion, or a change of status of the identity so that it can be undeleted if necessary. This might affect other identities. For example, deleting a User that is a member of a Group might cause a Modify event for that Group.
- **Rename:** An identity's naming attribute has changed.
- **Move:** An identity's logical location has changed. This usually applies to identities in a hierarchical system.

An event of a certain type in one system can result in an event of a different type in the synchronized system. For example, when a Modify event occurs for an identity that does not yet exist in the external system, an Add event is submitted.

There is one more type of event that does not represent a change but is a request for information: the Query event. NetIQ eDirectory issues queries to your application on the Subscriber channel. You can also query eDirectory from scripts on either Subscriber or Publisher channels.

NetIQ eDirectory supports all of the events above. You should make a list of what the result of a particular event in eDirectory will be in your external application. Conversely, you should list what event types can occur in your external application, which event types are relevant and what the result of relevant events should be in the Identity Vault.

The Connected System Schema File

The `schema.def` file on the connected system is stored in the `schema` directory under the driver installation directory. It is used to specify the classes and attributes that are available on the system.

The schema file is read by the driver shim when the Metadirectory engine requests it. This typically happens at driver startup. The schema file is also used by the Policy Editor to map the schema of the Identity Vault to the schema of the external application.

If you change the schema file, you must restart the driver shim and the driver.

The scripts written for the driver depend on the classes and attributes in the schema file.

Schema File Syntax

Each line in the schema file represents an element and must begin with the element name: SCHEMA, CLASS, or ATTRIBUTE.

The first element of the schema file is the schema definition. The schema definition is followed by class definitions. Each class definition can contain attribute definitions.

Except for the values of class and attribute names, the contents of the schema file are case insensitive.

Comments

Lines that begin with an octothorpe (#) are comments.

```
# This is a comment.
```

Schema Definition

The first line in the schema file that is not a comment must be the schema definition.

```
SCHEMA [HIERARCHICAL]
```

HIERARCHICAL specifies that the target application is not a flat set of users and groups, but is organized by hierarchical components, such as a directory-based container object.

Class Definition

```
CLASS className [CONTAINER]
```

You must specify a class name. Enclose the class name in double quotes (").

Add the CONTAINER keyword if objects of this class can contain other objects.

The class definition is ended by another class definition or by the end of the file.

Attribute Definition

Any number of attribute definitions can follow a class definition. Attribute definitions define attributes for the class whose definition they follow.

```
ATTRIBUTE attributeName [TypeAndProperties]
```

An attribute name is required. Enclose the attribute name in double quotes (").

If no attribute type is specified, the attribute has the string type. The allowable types are:

- STRING
- INTEGER
- STATE
- DN

The allowable attribute properties are:

- REQUIRED
- NAMING
- MULTIVALUED
- CASESENSITIVE
- READONLY

Example Schema File

```
SCHEMA HIERARCHICAL
   CLASS "User"
      ATTRIBUTE "cn" NAMING REQUIRED
      ATTRIBUTE "Group Membership" MULTIVALUED DN
   CLASS "Group"
      ATTRIBUTE "cn" NAMING REQUIRED
      ATTRIBUTE "Group Members" MULTIVALUED DN
```

The Connected System Include/Exclude File

You can use an optional include/exclude file on the connected system to control which identities are or are not synchronized between the Identity Vault and the connected system. Create a text file named `include-exclude.conf` and save it in the `conf` directory under your driver installation directory.

The file is read when the driver shim starts. If you make changes to it, you must restart the driver shim.

The include/exclude file can contain include rules and exclude rules. To ensure optimal performance, each include/exclude file should contain no more than 50 entries total.

You can use the include/exclude file to phase in your deployment of the Scripting driver, excluding most users and groups at first, and then adding more as you gain confidence and experience.

Topics in this section include

- "Include/Exclude Processing" on page 40
- "Include/Exclude File Syntax" on page 41
- "Example Include/Exclude Files" on page 44

Include/Exclude Processing

Identity Vault events for identities that match an exclude rule are discarded by the Subscriber shim. Connected system events for identities that match an exclude rule are not sent to the Metadirectory engine by the Publisher shim.

Included identities are treated normally by the Subscriber and Publisher shims.

Identities that do not match an include rule or an exclude rule in the file are included.

Identities are matched in the following priority:

1. Channel-specific (Publisher or Subscriber) exclude rules
2. Channel-specific include rules
3. General exclude rules
4. General include rules

Within each level of this matching priority, identities are matched against rules in the order that the rules appear in the file. The first rule that matches determines whether the identity is included or excluded.

Include/Exclude File Syntax

Except for class names, attribute names, and the values to match, the contents of the include/exclude file are case insensitive.

The include/exclude file can contain any number of include sections, exclude sections, and single-line rules.

Include sections and exclude sections can contain class matching rules, and class matching rules can contain attribute matching rules. Include sections and exclude sections can also contain association matching rules.

Include and exclude sections can be contained in subscriber and publisher sections to limit their scope to the specified channel.

Class and attribute names used in the include/exclude file must correspond to the names specified in the schema file. For details about the schema file, see "The Connected System Schema File" on page 38.

Comments

Lines that begin with an octothorpe (#) are comments.

```
# This is a comment.
```

Subscriber and Publisher Sections

Subscriber and publisher sections limit the include and exclude sections they contain to the specified channel.

A subscriber section begins with a subscriber line and ends with an `ENDSUBSCRIBER` line.

```
SUBSCRIBER
.
.
.
ENDSUBSCRIBER
```

A publisher section begins with a publisher line and ends with an `ENDPUBLISHER` line.

```
PUBLISHER
.
.
.
ENDPUBLISHER
```

Each subscriber and publisher section can contain include and exclude sections.

Include and Exclude Sections

Include and exclude sections provide rules to specify which objects are to be included or excluded from synchronization.

An include section begins with an include line and ends with an `ENDINCLUDE` line.

```
INCLUDE
.
.
.
ENDINCLUDE
```

An exclude section begins with an exclude line and ends with an `ENDEXCLUDE` line.

```
EXCLUDE
.
.
.
ENDEXCLUDE
```

You can use class matching rules and association matching rules within an include section and an exclude section.

Class Matching Rules

Use a class matching rule within an include section or an exclude section to specify the name of a class of objects to include or exclude.

A class matching rule is defined by a class line that specifies the name of the class and ends with an ENDCLASS line.

```
CLASS className
.
.
.
ENDCLASS
```

You can use attribute matching rules within a class matching rule.

Attribute Matching Rules

You can use attribute matching rules within a class matching rule to limit the objects that are included or excluded. If no attribute matching rules are specified for a class, all objects of the specified class are included or excluded.

An attribute matching rule comprises an attribute name, an equals sign (=), and an expression. The expression can be an exact value, or it can use limited regular expressions. For details about limited regular expressions, see “Limited Regular Expressions” on page 44.

```
attributeName=expression
```

Multiple attribute matching rules can be specified for a given class.

Attribute matching rules within a class matching rule are logically ANDed together. To logically OR attribute matching rules for a class, specify multiple class matching rules. For example, the following include/exclude file excludes both user01 and user02:

```
# Exclude the User object if its loginName is user01 or user02.
EXCLUDE
CLASS User
    loginName=user01
ENDCLASS
CLASS User
    loginName=user02
ENDCLASS
ENDEXCLUDE
```

Association Matching Rules

You can specify association matching rules in an include or exclude section. Association matching rule expressions can specify an exact association or a limited regular expression. For details about limited regular expressions, see “Limited Regular Expressions” on page 44.

The way associations are formed can be customized for an implementation. (See “Scripting Driver Data Definition” on page 35 for more information.)

This example works for associations that are a concatenation of the object name and class name. To exclude the root user, specify the following:

```
EXCLUDE
  rootUser
ENDEXCLUDE
```

Single-Line Rules

[SUBSCRIBER|PUBLISHER] INCLUDE|EXCLUDE [className] objectSelection

Where objectSelection can be

{associationMatch | attributeName=expression}

Single-line rules can specify the Subscriber or Publisher channel at the start of the rule. If a channel is specified, the rule applies only to that channel. Otherwise it applies to both channels.

You must specify whether the rule is to include or exclude the objects it matches.

You can specify a class name to limit matches to only objects of that class.

You must specify either an association or an attribute matching expression. The syntax of the association and attribute matching expression is the same as that of association matching rules and attribute matching rules previously described. For details, see "Association Matching Rules" on page 43 and "Attribute Matching Rules" on page 43.

For example, to ignore events from the ADMIN user in the Identity Vault, code:

```
# Do not subscribe to events for the ADMIN user.
SUBSCRIBER EXCLUDE adminUser
```

Limited Regular Expressions

A limited regular expression is a pattern used to match a string of characters.

Character matching is case sensitive.

Any literal character matches that character.

A period (.) matches any single character.

A bracket expression is a set of characters enclosed by left ([) and right (]) brackets that matches any listed character. Within a bracket expression, a range expression is a pair of characters separated by a hyphen, and is equivalent to listing all of the characters that sort between the given characters, inclusive. For example, [0-9] matches any single digit.

An asterisk (*) indicates that the preceding item is matched zero or more times.

A plus sign (+) indicates that the preceding item is matched one or more times.

A question mark (?) indicates that the preceding item is matched zero or one times.

You can use parentheses to group multiple expressions into a single item. For example, (abc)+ matches abc, abcabc, abcabcabc, etc. Nesting of parentheses is not supported.

Example Include/Exclude Files

- "Example 1" on page 44
- "Example 2" on page 45
- "Example 3" on page 45

Example 1

```
# Exclude users whose names start with temp
EXCLUDE
    CLASS User
        loginName=temp.*
    ENDCLASS
ENDEXCLUDE
```

Example 2

```
# Exclude usera and userb
# Because attribute rules are ANDed, these must be in separate
# CLASS sections.
EXCLUDE
    CLASS User
        loginName=usera
    ENDCLASS
    CLASS User
        loginName=userb
    ENDCLASS
ENDEXCLUDE
```

Example 3

```
# Exclude all users except those whose names start with idm
# This works because channel-specific matching takes precedence
# over general matching.
EXCLUDE
    CLASS User
    ENDCLASS
ENDEXCLUDE

SUBSCRIBER INCLUDE User loginName=idm.*
PUBLISHER INCLUDE User loginName=idm.*
```

Managing Additional Attributes

You can add additional attributes to the driver for both the Publisher and Subscriber channels. These attributes can be accessed by the scripts for all event types.

To publish or subscribe to additional attributes, you must add them to the filter and add support for them into the scripts. Topics in this section include

- "Modifying the Filter" on page 45
- "Modifying the Scripts for New Attributes" on page 46

Modifying the Filter

1. Select the Identity Manager **Drivers** module from the Identity Console landing page.
2. Select the **Data Transformation and Synchronization** tab.
3. Expand the **Class Attribute Filters** panel and select a class name.
4. Click **Add Attribute,** then select the attribute from the list.
5. Select the flow of this attribute for the Publisher and Subscriber channels.
    - **Synchronize:** Changes to this object are reported and automatically synchronized.
    - **Ignore:** Changes to this object are not reported and not automatically synchronized.
    - **Notify:** Changes to this object are reported, but not automatically synchronized.

- **Reset:** Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher or Subscriber channel, but not both.)

6. Click the **Save** icon.
7. If you want to map this attribute to an existing attribute in the connected system's schema, modify the Schema Mapping policy for the driver.

For complete details about managing filters and Schema Mapping policies, see Policies in the *Identity Manager Guide*.

Modifying the Scripts for New Attributes

In the Subscriber channel, scripts are called to take the appropriate action for each type of event. You will need to modify the appropriate scripts to read the values from the new attributes.

Publishing additional attributes requires that you act on changes made in the connected system application. In addition, the schema.def file should be updated with the additional attributes (see "The Connected System Schema File" on page 38).

UNIX Shell Developer Guide

The Scripting driver provides a complete Shell script API for interacting with identity management systems whose tools (including APIs) are available on Linux and UNIX. The Identity Vault and Identity Manager can run on any supported operating system. Identity Manager communicates with the driver on a different system over an encrypted network connection.

Before beginning script development, review the preceding topics in this section for information on defining what data will be synchronized between identity management systems.

With additional development work, the driver can also be customized to support any scripting language that supports command-line operation.

Developing a custom driver with Shell scripts is discussed in the following topics:

- "Application Tools Evaluation" on page 46
- "Policy and Script Development" on page 47
- "Deployment" on page 60

Application Tools Evaluation

To change the data in your external application, you need to know how to use the application's tools or API (Application Programming Interface). These tools must provide automated operation and not require user input.

Application Command Line Tools

An application often provides command line tools. These tools are manually executed from a shell, and they can be executed from scripts. Suppose the application provides a tool to add identities with a program called appadd.

```
appadd -n "Bob Smith" -t "818-555-2100"
```

This command adds an identity named “Bob Smith” with the specified phone number. The strings following the program name are called parameters or arguments. The Linux and UNIX Scripting driver provides a function called EXEC to execute external programs, log the command to the system log and produce a status document indicating the level of success.

```
CommandLine="appadd -n $UserName -t $PhoneNumber"
EXEC $CommandLine
```

For command line tools, you can construct the command line’s parameters using the values passed to the script, then execute the program.

Application Event Monitoring

You also need to determine what tools are available for monitoring event changes in the application. The Scripting driver works on a polling system. It periodically calls a polling script to determine what has changed in the external application. You can use the following ideas for monitoring changes:

- The first time the polling script is run, a list of identities and relevant attributes is read from the application using an application-provided tool. This list is stored as a file. On subsequent polls, a new list is generated and compared to the old list. Any differences are submitted as events to the driver.
- The application provides a tool that allows you to request all identities that have changed after a certain point in time. The polling script requests events that have occurred since the previous poll.
- The application allows a script to be run when an event occurs. You can write a script that stores the event data into a file. When the Script driver polling script runs, it consumes this file and submits the data as an event to the driver using the change log tool, `usclh`. For detailed information on `usclh`, see “Publisher Change Log Tool” on page 175

Monitoring the application’s changes might be the most difficult aspect of developing your driver. You must study your application’s tools to determine the best way to achieve synchronization.

Policy and Script Development

At this point you should have a list of what data will be synchronized, how events will be handled and what application tools are available. It is time to develop the heart of your driver in policies and scripts.

Many types of tasks can be handled in driver policies. You can import the driver configuration provided with the Scripting driver, and then edit policies in NetIQ Identity Console. You can also edit policies and simulate their operation in NetIQ Designer. The extensive functionality of policies is outside the scope of this document, so you should refer to the appropriate publications at the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).

It’s often difficult to write complex tasks inside policies, such as executing external commands, processing input and output, and file I/O. Tasks requiring such operations are better suited in scripts, where an entire language environment and tools are available. You can also accomplish many of the operations performed in policies, so if you are more familiar with your scripting language than policies, you can develop your driver more quickly by using scripts. Scripting languages such as Perl and Shell scripts offer an environment that is often well suited for your target application's APIs or developer kits. For example, your target application might already contain Perl library routines for manipulating the application's identities.

Event Data Format

Windows Event Data

Event data is submitted to the scripts in name/value pair format. This format consists of lines containing a name, an equal sign (=) and a value. Therefore each line is a name/value pair. Each name/value pair is unique, but there can be multiple name/value pairs with identical names but different values.

```
ASSOCIATION=BobUser
ADD_TELEPHONE=818-555-2100
ADD_TELEPHONE=818-555-9842
```

You typically don't need to worry about the format. The script library provides functions for retrieving event data.

Linux and UNIX Event Data

On Linux and UNIX installations, event data is provided to the script framework, by the driver shim, using IPC Shared Memory. The document is stored in structured, binary format, which can be parsed using the Shared Memory Helper tool (`ussmh`). This tool is abstracted by some of the languages to make reading data easier. However, if you are using Shell Scripts for reading data, it's important to understand name/value pairs.

The name/value pairs format allow Shell Scripts to get data and store them in Shell variables, so they can be directly used by Shell Scripts. The `IDMGETVAR` shell function invokes the `ussmh` tool and captures the standard output to store away the value for the named variable. Some named variables, such as READ_ATTRS, SEARCH_ATTRS, SEARCH_CLASSES and many custom attributes may be multi-valued. For multi-valued attributes, the value will presented as a newline-delimited value and must be parsed using shell techniques such as `IFS` (internal field separator).

Subscriber Script Development

After all Policy processing is complete, Identity Manager submits the event in XML format to the driver shim. The driver shim submits the event data to the scripts.

In the default Scripting driver, the `subscriber.sh` script in the `scripts` folder is called. This script does some preliminary processing, and then calls a routine from an included script. The included scripts correspond to the Subscriber event types: `add.sh`, `modify.sh`, `modify-password.sh`, `delete.sh`, `rename.sh`, `move.sh`, and `query.sh`.

For each event type, you should retrieve the information you need from the event data, submit changes to the external application using application-provided tools, and return a status (such as success or failure) to Identity Manager.

Event data is retrieved primarily using the IDMGETVAR function. For detailed information on how to use IDMGETVAR, see "UNIX Shell (idmlib.sh) Reference" on page 149.This function returns an array of values corresponding to the name specified as the function's parameter. The following table shows many item names.

Table 5-1 Items

Name	Description
COMMAND	The command for the event, usually indicating the event type. Possible values are: `add`, `modify`, `delete`, `rename`, `modify-password`, `check-object-password`.
ASSOCIATION	The identifier that distinguishes an identity on both identity management systems.
CLASS_NAME	An identity's class, such as User or Group.
SRC_DN	An identity's distinguished name (DN) in the namespace of the source (sender), in slash format.
EVENT_ID	An identifier for the event, for internal use.
SRC_ENTRY_ID	An identifier for the identity that generated the event, in the namespace of the source (sender).
DEST_DN	An identity's distinguished name (DN) in the namespace of the destination (receiver), in slash format.
DEST_ENTRY_ID	An identifier for an entry in the namespace of the destination (receiver).
ADD_*attr_name*	A value to be added to an identity, for attribute *attr_name*.
REMOVE_*attr_name*	A value to be removed from an identity, for attribute *attr_name*.
ADD_REF_*attr_name*	A value to be added to attribute *attr_name*, where the value is an association referring to another identity.
REMOVE_REF_*attr_name*	A value to be removed from attribute *attr_name*, where the value is an association referring to another identity.
REMOVE_ALL_attr_name	Indicates that all values for the attribute *attr_name* should be removed.
OLD_PASSWORD	The previous password for an identity that has changed its password. Used in Modify Password events.
PASSWORD	The new password for an identity. Used in Add and Modify Password events.
OLD_SRC_DN	The distinguished name of an identity before a Move or Rename event.
REMOVE_OLD_NAME	Specifies whether an old relative distinguished name should be deleted or retained. Used in Rename events.
STATUS_LEVEL	The status of an event: success, warning, retry, error or fatal.
STATUS_MESSAGE	A message to report with a status.
STATUS_TYPE	A type of status, such as heartbeat.

Examples Of Obtaining Event Data

Example 1:

```
command='IDMGETVAR "COMMAND"'
# check for an add event
if [ "$command" = "add" ]; then
  # call the add script
  add.sh
fi
```

Example 2:

```
# obtain the event's association and CN attribute
ASSOCIATION=‘IDMGETVAR "ASSOCIATION"‘
CN='IDMGETVAR "ADD_CN"'
if [ "$CN" = "bob" ]; then
  # for "bob", check to see if he's been enabled
  ENABLE=‘IDMGETVAR "REMOVE_Login Disabled"‘
  if [ "$ENABLE" = "true" ]; then
    # bob is enabled again
    cmd="appenable -association $ASSOCIATION"
    EXEC "$cmd"
  fi
fi
```

Handling Associations

The association value indicates which identity has been changed. If the identity has no association, an association must be generated for it using an implementation-specific rule that you have adopted. When Identity Manager processes an event for an identity with no association, it executes the driver’s Matching policy. This policy attempts to match the event’s identity to an identity on the external application’s system. Usually doing this involves executing a query. The default Matching policy included with the Scripting driver queries for matching Users and Groups based on the CN attribute. If the event’s identity matches an identity on the external application, both identities must be assigned the new association. Assigning this association can be done as part of the query-handling script. (Handling queries is discussed in more detail in “Handling Query Events” on page 52.) If no identity matches, an Add event is issued, and the new association can be assigned as part of the Add event-handling script:

```
# Adding an association
IDMSETVAR "COMMAND" "ADD_ASSOCIATION"
IDMSETVAR "ASSOCIATION" "$MyAssociation"
IDMSETVAR "EVENT_ID" "$EVENT_ID"
IDMSETVAR "DEST_DN" "$SRC_DN"
IDMSETVAR "DEST_ENTRY_ID" "$SRC_ENTRY_ID"
```

The above example demonstrates each name/value pair that must be set for an association to be assigned by the Identity Manager engine. The values of `EVENT_ID`, SRC_DN and `SRC_ENTRY_ID` are always sent by the engine during an add event, and therefore, are available for your add script to obtain using IDMGETVAR. The example above also illustrates the IDMSETVAR function. For detailed information on how to use IDMSETVAR, see “UNIX Shell (idmlib.sh) Reference” on page 149. This function sets a name and value which indicates what action Identity Manager should perform. For example, the pair `COMMAND` and `ADD_ASSOCIATION` instructs the shim to create an add-association document to assign an association to an identity, as discussed above. The pair `EVENT_ID` and `$EVENT_ID` instruct the shim to assign add-association document an event-id described by the variable `$EVENT_ID`. This is important for the engine to match documents sent and returned on the subscriber channel.

The Subscriber can also issue `MODIFY_ASSOCIATION` and `REMOVE_ASSOCIATION` commands:

```
# Removing an association
IDMSETVAR "COMMAND" "REMOVE_ASSOCIATION"
IDMSETVAR "ASSOCIATION" "$MyAssociation"
IDMSETVAR "EVENT_ID" "$EVENT_ID"
IDMSETVAR "DEST_DN" "$SRC_DN"
IDMSETVAR "DEST_ENTRY_ID" "$SRC_ENTRY_ID"

# Modifying an association
IDMSETVAR "COMMAND" "MODIFY_ASSOCIATION"
IDMSETVAR "ASSOCIATION" "$OldAssociation"
IDMSETVAR "ASSOCIATION" "$NewAssociation"
IDMSETVAR "EVENT_ID" "$EVENT_ID"
IDMSETVAR "DEST_DN" "$SRC_DN"
IDMSETVAR "DEST_ENTRY_ID" "$SRC_ENTRY_ID"
```

Returning an Event Status

On the Subscriber channel, you often do not need Identity Manager to perform an action, but simply need to report a status. The STATUS_ subroutines noted below can be used to indicate a status to Identity Manager. They take a message to be logged as their parameter.

Table 5-2 *Status Subroutines*

Subroutine	Identity Manager Action
STATUS_SUCCESS	Identity Manager marks the event as a success and submits the next event in the event queue, if any. You should issue this status even if your script does nothing.
STATUS_WARNING	The event can be processed, but it might require attention. Identity Manager issues your warning message in its log, and then submits the next event.
STATUS_RETRY	The event cannot be processed, but Identity Manager should resubmit the event because it should be able to be processed soon. This status can be issued if your external application appears to be temporarily unavailable. However, this status should be used cautiously because a backlog results if Identity Manager continually retries one event.
STATUS_ERROR	The event cannot be processed and it should not be resubmitted. Identity Manager issues the error message and submits the next event. You should make a detailed error message so the problem can be corrected.
STATUS_FATAL	For some reason, the driver must be stopped. Identity Manager issues your message and stops the driver. This could be used if the external application appears to be permanently offline. The event remains in the queue and is resubmitted when the driver is restarted.

Examples Using the STATUS Functions

Example 1:

```
EXEC "$cmd"
if [ $? -eq 0 ]; then
  STATUS_SUCCESS "Command was successful"
fi
```

Example 2:

```
EXEC "$cmd"
if [ $? -eq 0 ]; then
  if [ -z "$password" ]; then
    # created, but no password
    STATUS_WARNING "User added without password"
  fi
fi
```

Example 3:

```
EXEC "$cmd"
if [ $? -ne 0 ]; then
  STATUS_ERROR "Command failed"
fi
```

Writing Values

IDMSETVAR is used to set values to return to Identity Manager. For detailed information on how to use IDMSETVAR, see "UNIX Shell (idmlib.sh) Reference" on page 149. It is passed a name and value. In the previous ADD_ASSOCIATION example, IDMSETVAR is used to set the ASSOCIATION value. You can specify values for items listed in the table above. Generally, the only time IDMSETVAR is used is to add, modify and delete associations or return information for a query operation. Other information returned to the shim by the scripts is done through other command functions, such as STATUS_SUCCESS, which use IDMSETVAR indirectly.

Handling Query Events

For Query events, Identity Manager submits values that define the parameters of a search of the external application's identity management system. Queries are usually issued from the Policies you have defined for your system. The table below specifies values that can be specified in queries. Not all values are relevant to your external application.

Table 5-3 *Values for Queries*

Value Name	Description
SCOPE	Specifies what identities will be searched. A base object is specified with the ASSOCIATION or DEST_DN values (see below). The value "entry" means that only the base object is searched. The value "subordinates" means that the immediate subordinates of the base object are searched. The value "subtree" (the default) indicates that the base object and all subordinates are searched. The last two values are only relevant in a hierarchical system.
ASSOCIATION	The base object for the search. If both ASSOCIATION and DEST_DN have values, ASSOCIATION is used. If neither is specified, the base object is the root of the identity management system.

Value Name	Description
DEST_DN	The base object for the search (see also ASSOCIATION above).
CLASS_NAME	The base class of the base object.
EVENT_ID	An identifier for the event.
SEARCH_CLASSES	A list of classes for which to search. Only identities of these classes are returned. If not specified, all identities in the scope matching SEARCH_ATTR_ values are returned (see below).
SEARCH_ATTRS	A list of the attribute names specified in SEARCH_ATTR_ values (see below).
SEARCH_ATTR_*attr_name*	A value that the specified attribute must match. Replace *attr_name* with the desired attribute name. Only identities matching all SEARCH_ATTR_ filters are returned.
READ_ATTRS	A list of the attribute names whose values are returned for each matching identity.
ALL_READ_ATTRS	The presence of this value indicates that all attribute values should be returned for matching identities.
NO_READ_ATTRS	The presence of this value indicates that no attribute values should be returned for matching identities.
READ_PARENT	The presence of this value indicates that the parent object of each matching identity should be returned. Only relevant in hierarchical systems.

Execute the query against the external application using application-provided tools. Then return each identity by setting an INSTANCE command, followed by relevant values from the list below.

Table 5-4 *Query Instance Values*

Value Name	Description
CLASS_NAME	The class of the identity. Required.
SRC_DN	A distinguished name representing the logical location of the identity in the system (optional).
ASSOCIATION	The association of the identity, if available (optional).
PARENT	The association of the parent object of the identity (optional). Only relevant in hierarchical systems.
ATTR_*attr_name*	A list of values for the attribute specified by *attr_name*. Return attribute values specified by the READ_ATTRS value.

After returning all identities, call STATUS_SUCCESS to indicate a successful query.

Subscriber Summary and Examples

Below is a more detailed summary of the actions to take for a non-Query event.

1. Gather information about the event using IDMGETVAR. Return a warning or error if there is a problem.
2. Submit the event data to the external application using application-provided tools.
3. Set return values with IDMSETVAR.
4. If you have not already done so, set a status with a STATUS_ *subroutine*.

Below is an example add.sh, which forms an association from an identity's CN and class name, and uses a hypothetical tool called appadd.

```
#!/bin/sh
ClassName=`IDMGETVAR "CLASS_NAME"` CN=`IDMGETVAR "CN"`
EVENT_ID=`IDMGETVAR "EVENT_ID"`
SRC_DN=`IDMGETVAR "SRC_DN"`
SRC_ENTRY_ID=`IDMGETVAR "SRC_ENTRY_ID"`
PhoneNumber=`IDMGETVAR "Telephone"`

if [ -z "$ClassName" -o -z "$CN" ]; then
  STATUS_ERROR "Add event: missing CLASS_NAME and/or CN"
else
  Command="appadd -n $CN -t $PhoneNumber"
  EXEC $Command
  if [ $? -eq 0 ]; then
    IDMSETVAR "COMMAND" "ADD_ASSOCIATION"
    IDMSETVAR "ASSOCIATION" $CN $ClassName
    IDMSETVAR "EVENT_ID" "$EVENT_ID"
    IDMSETVAR "DEST_DN" "$SRC_DN"
    IDMSETVAR "DEST_ENTRY_ID" "$SRC_ENTRY_ID"

    STATUS_SUCCESS "Add event succeeded"
  else
    STATUS_ERROR "Add event failed with error code $RC"
  fi
fi
```

Handling a query is a similar process, except that you return INSTANCE items rather than using other commands. Below is an example `query.sh` that searches an external application for a telephone number. It uses a hypothetical tool called `appsearch`.

```
#!/bin/sh
SearchName=`IDMGETVAR "SEARCH_ATTR_CN"`
EVENT_ID=`IDMGETVAR "EVENT_ID"`
ASSOCIATION=`IDMGETVAR "ASSOCIATION"`
CLASS_NAME=`IDMGETVAR "CLASS_NAME"`

if [ -z "$SearchName" ]; then
  STATUS_ERROR "Query: no search value"
else
  Command="appsearch -n $SearchName"
  Results=`$Command`
  if [ -n "$Results" ]; then
  Phone=`echo $Results | awk '{print $1}'`
  IDMSETVAR "COMMAND" "INSTANCE"
  IDMSETVAR "EVENT_ID" "$EVENT_ID"
  IDMSETVAR "CLASS_NAME" "$CLASS_NAME"
  IDMSETVAR "ASSOCIATION" "$ASSOCIATION"
  IDMSETVAR "ATTR_Telephone" "$Phone"

  STATUS_SUCCESS "Query succeeded"
  else
    # Return success with no results
    STATUS_SUCCESS "Query succeeded (no matches)"
  fi
fi
```

Publisher Script Development

Events that occur on the external application are submitted to Identity Manager on the Publisher channel. The Scripting driver periodically polls the external application for events. How this poll detects events is implementation-specific and must be defined by you.

Polling for Application Events

The Driver calls `poll.sh` to detect application events. Implement `poll.sh` as follows:

1. Use application-provided tools to detect events in your application, as discussed in Step 2.
2. For each event, call the changelog tool `usclh` to submit the event to be published. The changelog tool allows for additional information to be supplied through standard input. This is an appropriate mechanism for passing data that might be too large for command line or too sensitive to appear in a shell's history or environment. For detailed information on `usclh`, see "Publisher Change Log Tool" on page 175.

The following is an example of a `poll.sh` that checks for a password change. It uses a hypothetical application tool called appchg.

```
#!/bin/sh
# look for password changes
Results=`appchg --passwd-changes`
for Result in $Results; do
  # Entries are in the format "association:password"
  Association=`echo $Result | awk -F: '{print $1}'`
  Password=`echo $Result | awk -F: '{print $2}'`

  # submit a password change event
  usclh -t modify-password -a $Association <<EOF
$Password
EOF
done

# look for attribute values being added to each user
Results=`appchg --add-attr-changes`
for Result in $Results; do
  # Entries are in the format "association:attribute:value"
  Association=`echo $Result | awk -F: '{print $1}'`
  Attribute=`echo $Result | awk -F: '{print $2}'`
  Value=`echo $Result | awk -F: '{print $3}'`

  # submit the added attribute value
  usclh -t modify -c User -a $Association <<EOF
ADD_$Attribute=$Value
EOF
done

# look for attribute values being removed from each user
Results=`appchg --remove-attr-changes`
for Result in $Results; do
  # Entries are in the format "association:attribute:value"
  Association=`echo $Result | awk -F: '{print $1}'`
  Attribute=`echo $Result | awk -F: '{print $2}'`
  Value=`echo $Result | awk -F: '{print $3}'`

  # submit the removed attribute value
  usclh -t modify -c User -a $Association <<EOF
REMOVE_$Attribute=$Value
EOF
done
```

In the above example, three separate events are submitted to the publisher change log using the change log tool, usclh. The first invocation submits a modify-password event to be published. The second event submits a modify event to be published for an attribute add. The third invocation submits another modify event to be published for an attribute removal. The second and third invocations can be combined into a single modify event, if desired.

Events submitted using `usclh` are processed through your driver's Publisher channel policies. See your Identity Manager 4.8 Policy guides (https://www.netiq.com/documentation/idm45/) for more information.

Using the Heartbeat Script

Another script executed in the Publisher Channel is `heartbeat.sh`. This script is executed when the Publisher Channel is idle for the interval specified in the Driver parameters. (You can set the interval to 0 so no heartbeat is issued.) You can use the heartbeat to check the availability of the external system or do “idle state” tasks. The HEARTBEAT_SUCCESS, HEARTBEAT_WARNING, and HEARTBEAT_ERROR subroutines can be used to indicate the result of the heartbeat. Below is an example based on a hypothetical tool called `apphealth`.

```
apphealth
RC=$?
if [ $RC -eq 0 ]; then
  HEARTBEAT_SUCCESS "Heartbeat succeeded"
else
  HEARTBEAT_ERROR "Heartbeat failed with error code $RC"
fi
```

The response to the heartbeat is implementation-dependent, and can be defined in Policies or in the script itself. You could send a message to auditing using NetIQ Audit. You could store a value in a file, and have Subscriber scripts read the file and call STATUS_RETRY if they find that value in the file.

Other Scripting Topics

- “Driver Parameters” on page 57
- “Querying the Identity Vault” on page 58
- “Tracing and Debugging” on page 58

Driver Parameters

A driver has values known as driver parameters. The driver parameters are divided into driver settings applicable to the whole driver, and Subscriber and Publisher settings for their respective channels. The IDMGETDRVVAR, IDMGETSUBVAR, and IDMGETPUBVAR functions can be used to retrieve these values. The table below shows parameters in the default Scripting driver. Other parameters can be added to the driver’s XML configuration file see “Managing Identity Manager Drivers” in the *Identity Manager Administration Guide*).

Table 5-5 *Scripting Driver Parameters*

Parameter Name	Driver/Channel	Description	Values
INSTALL_PATH	Driver	The installation path of the Driver	string value
auto-loopback-detection	Driver	Whether to enable automatic loopback detection	true/false
subscriber-script	Subscriber	The root script file for Subscriber events, relative to the Driver installation path	string value

Parameter Name	Driver/Channel	Description	Values
pub-polling-interval	Publisher	The interval in seconds between Publisher polls for application events	number
pub-heartbeat-interval	Publisher	The amount of idle time in seconds before a heartbeat event is issued	number
pub-disabled	Publisher	Whether the Publisher Channel (i.e., polling) is disabled	true/false

In the following example, a script retrieves the Publisher polling interval.

```
PollingInterval=`IDMGETPUBVAR "pub-polling-interval"`
```

Querying the Identity Vault

Scripts might need to retrieve information from the Identity Vault. They can do this by issuing a query.

1. Execute the query by calling IDMQUERY with the appropriate parameters:
   - The first parameter is the class-name
   - The second parameter is the association of the object to query
   - The third parameter are the attributes to read, comma-separated
2. Read the result (instance) using IDMGETQVAR.

Query support is currently limited. It only returns one instance based on the specified association or DN. (If both association and DN are specified, association is used.) The functions below allow you to retrieve information from the instance.

The following is an example of a query of the Identity Vault that retrieves the address and ZIP code for user Bob.

```
IDMQUERY "User" "Bob" "SA,Postal Code"

Address=`IDMGETQVAR "SA"`
ZIPCode=`IDMGETQVAR "Postal Code"`
# ... etc. ...
```

Tracing and Debugging

The IDMTRACE function allows you to write a message to the Trace Log. Tracing is useful for script debugging and auditing.

```
IDMTRACE "Trace message"
```

When developing scripts, you might need to do some debugging to track down problems. The following list indicates some facilities for debugging.

- The Driver traces activity to its Trace file (`logs/trace.log` by default). The trace level setting in `conf/usdrv.conf` controls how much debugging is written to the log.

Trace Level	Description
0	No debugging.
1-3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous level plus Remote Loader, driver, driver shim, and Driver connection messages.
5-7	Previous level plus Change Log and loopback messages. Higher trace levels provide more detail.
8	Previous level plus Driver status log, Driver parameters, Driver command line, Driver security, Driver Web server, Driver schema, Driver encryption, Driver SOAP API, and Driver include/exclude file messages.
9	Previous level plus low-level networking and operating system messages.
10	Previous level plus maximum low-level program details.

The trace level is set using the -trace option in `usdrv.conf`, for example -trace 9.

You can view the trace file through a Web browser:

1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
2. Authenticate by using any username and the password that you specified as the Remote Loader password.
3. Click **Trace**.

- The IDMTrace function described above writes output to the trace file specified in the Driver Parameters (`logs/script-trace.log` by default).
- The eDirectory tool DSTrace can be used to monitor Identity Manager activity. Set the tracing level for the driver in Identity Console. DSTrace shows the XML documents being submitted to the driver for events, and how Policies are evaluated. It also shows the status and message for each event.
- The Status Log is written to `logs/dirxml.log`. It shows a summary of the events that have been recorded on the Subscriber and Publisher channels.

  You can view the Status Log through a Web browser:

  1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
  2. Authenticate by using any username and the password that you specified as the Remote Loader password.
  3. Click **Status**.

Although it is best to start the driver in production environments from the startup script, you can run usdrv manually. When you do so, any text written to standard output from scripts is displayed in the interactive shell.

Deployment

The Scripting driver is installed by using a setup program. See "Installing the Linux and UNIX Scripting Driver Shim" on page 19 for more information on installing the default driver.

Deploying a Custom Driver

To deploy your custom driver, the end user should first run the Scripting driver installation program provided by the installation media ("Installing the Linux and UNIX Scripting Driver Shim" on page 19). This program installs core files needed by the driver. Then, your custom driver files can be deployed in any convenient way, whether through an installation program or even simply an archive file. The table below shows the directory structure below the installation directory and what files are installed.

Table 5-6 *Installation Directories and Files*

Directory	Description	Required Files

`bin/`	Location of executable programs	`usdrv`
		`ussmh`
		`usclh`

`changelog/`	Used for Publisher event processing	None
`conf/`	Location of the driver shim configuration file	usdrv.conf (customized)
`keys/`	Location of security key files	None
`logs/`	Location of log files	None
`loopback/`	Used for automatic loopback detection	None
`rules/`	Location of Driver configuration file	`Scripting.xml` (customized)
`schema/`	Location of schema files	`schema.def` (customized)
`scripts/`	Location of script files	

On Linux and UNIX, the Scripting driver is installed to `/opt/novell/usdrv`.

The formats of `usdrv.conf` and `schema.def` can be viewed in "The Driver Shim Configuration File" on page 32 and "The Connected System Schema File" on page 38.

If SSL encryption is desired for communication between the driver shim and Identity Manager engine, a certificate must be retrieved from the Identity Vault. Run `usdrv -s` and follow the prompts to retrieve the certificate, which will be stored in the keys/ directory. You must have LDAP with SSL available for the Metadirectory. When making an installation program for deployment, you might want to run `usdrv -s` as part of the installation.

To ensure that only authorized systems access the Metadirectory, a Driver Object Password and Remote Loader Password are used. Run `usdrv -sp` and enter the passwords at the prompts. This action can be incorporated into an installation program.

You should distribute the XML configuration file that contains parameters and policies your Driver needs. The user can then select it when installing your Driver.

Perl Developer Guide

The Scripting driver provides a complete Perl API for interacting with identity management systems whose tools (including APIs) are available on Linux and UNIX. The Identity Vault and Identity Manager can run on any supported operating system. Identity Manager can communicate with any supported system on which the driver is installed via an encrypted network connection.

Before beginning script development, review the preceding topics in this section for information on defining what data is synchronized between identity management systems.

With additional development work, the driver can also be customized to support any scripting language that supports command-line operation.

Developing a custom driver with Perl scripts is discussed in this section. Topics include

- "Application Tools Evaluation" on page 61
- "Policy and Script Development" on page 62
- "Deployment" on page 74

Application Tools Evaluation

To change the data in your external application, you need to know how to use the application's tools or API (Application Programming Interface). These tools must provide automated operation and not require user input.

Application Command Line Tools

An application often provides command line tools. These tools are manually executed from the command line, and they can be executed from scripts. For example, suppose the application provides a tool to add identities with a program called appadd.

```
appadd -n "Bob Smith" -t "818-555-2100"
```

This command adds an identity named "Bob Smith" with the specified phone number. The strings following the program name are called parameters or arguments. The Linux and UNIX Scripting driver provides a function called exec to execute external programs, log the command to the system log, and produce a status document indicating the level of success.

```
$CommandLine="appadd -n $; UserName -t $PhoneNumber";
$idmlib = new IDMLib();
$idmlib->exec($CommandLine);
```

For command line tools, you can construct the command line's parameters using the values passed to the script, then execute the program.

Application Event Monitoring

You also need to determine what tools are available for monitoring event changes in the application. The Scripting driver works on a polling system. It periodically calls a polling script to determine what has changed in the external application. You can use the following ideas for monitoring changes:

- The first time the polling script is run, a list of identities and relevant attributes is read from the application using an application-provided tool. This list is stored as a file. On subsequent polls, a new list is generated and compared to the old list. Any differences are submitted as events to the driver.
- The application provides a tool that allows you to request all identities that have changed after a certain point in time. The polling script requests events that have occurred since the previous poll.
- The application allows a script to be run when an event occurs. You can write a script that stores the event data into a file. When the Script driver polling script runs, it consumes this file and submits the data as an event to the driver using the `usclh` change log tool. For detailed information on `usclh`, see "Publisher Change Log Tool" on page 175.

Monitoring the application's changes might be the most difficult aspect of developing your driver. You must study your application's tools to determine the best way to achieve synchronization.

Policy and Script Development

At this point you should have a list of what data will be synchronized, how events will be handled, and what application tools are available. It is time to develop the heart of your driver in policies and scripts.

Many types of tasks can be handled in driver policies. You can import the driver configuration provided with the Scripting driver, and then edit policies in NetIQ Identity Console. You can also edit policies and simulate their operation in NetIQ Designer. The extensive functionality of policies is outside the scope of this document, and so you should refer to the appropriate policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for help.

It's often difficult to write complex tasks inside policies, such as executing external commands, processing input and output, and file I/O. Tasks requiring such operations are better suited in scripts, where an entire language environment and tools are available. You can also accomplish many of the operations performed in policies, so if you are more familiar with your scripting language than policies, you can develop your driver more quickly by using scripts. Scripting languages such as Perl and Shell scripts offer an environment that is often well suited for your target application's APIs or developer kits. For example, your target application might already contain Perl library routines for manipulating the application's identities.

Event Data Format

Event data is submitted to the scripts in name/value pair format. This format consists of lines containing a name, an equal sign (=) and a value. Therefore, each line is a name/value pair. Each name/value pair is unique, but there can be multiple name/value pairs with identical names but different values.

```
ASSOCIATION=BobUser
ADD_TELEPHONE=818-555-2100
ADD_TELEPHONE=818-555-9842
```

You typically don't need to worry about the format. The script library provides functions for retrieving event data.

Subscriber Script Development

After all Policy processing is complete, Identity Manager submits the event in XML format to the driver shim. The driver shim submits the event data to the scripts.

In the default Scripting driver, the `subscriber.pl` script in the `scripts` folder is called. This script does some preliminary processing, and then calls a routine from an included script. The included scripts correspond to the Subscriber event types: `add.pl`, `modify.pl`, `modify-password.pl`, `delete.pl`, `rename.pl`, `move.pl`, and `query.pl`.

For each event type, you should retrieve the information you need from the event data, submit changes to the external application using application-provided tools and return a status (such as success or failure) to Identity Manager.

Event data is retrieved primarily by using the `$idmlib->idmgetvar()` function. This function returns an array of values corresponding to the name specified as the function's parameter. The following table shows many item names.

Table 5-7 *Item Names*

Name	Description
COMMAND	The command for the event, usually indicating the event type. Possible values are: `add`, `modify`, `delete`, `rename`, `modify-password`, `check-object-password`.
ASSOCIATION	The identifier that distinguishes an identity on both identity management systems.
CLASS_NAME	An identity's class, such as User or Group.
SRC_DN	An identity's distinguished name (DN) in the namespace of the source (sender), in slash format.
EVENT_ID	An identifier for the event, for internal use.
SRC_ENTRY_ID	An identifier for the identity that generated the event, in the namespace of the source (sender).
DEST_DN	An identity's distinguished name (DN) in the namespace of the destination (receiver), in slash format.
DEST_ENTRY_ID	An identifier for an entry in the namespace of the destination (receiver).
ADD_{attr_name}	A value to be added to an identity, for attribute {attr_name}.
REMOVE_{attr_name}	A value to be removed from an identity, for attribute {attr_name}.

Name	Description
ADD_REF_{attr_name}	A value to be added to attribute {attr_name}, where the value is an association referring to another identity.
REMOVE_REF_{attr_name}	A value to be removed from attribute {attr_name}, where the value is an association referring to another identity.
OLD_PASSWORD	The previous password for an identity that has changed its password. Used in Modify Password events.
PASSWORD	The new password for an identity. Used in Add and Modify Password events.
OLD_SRC_DN	The distinguished name of an identity before a Move or Rename event.
REMOVE_OLD_NAME	Specifies whether an old relative distinguished name should be deleted or retained. Used in Rename events.
STATUS_LEVEL	The status of an event: success, warning, retry, error or fatal.
STATUS_MESSAGE	A message to report with a status.
STATUS_TYPE	A type of status, such as heartbeat.

Examples Of Obtaining Event Data

Example 1:

```
my $idmlib = new IDMLib();
my $command = $idmlib->idmgetvar("COMMAND");
# check for an add event
if ($command eq "add") {
  # call the add script
  do add.pl;
}
```

Example 2:

```
my $idmlib = new IDMLib();
# obtain the event's association and CN attribute
my $association = $idmlib->idmgetvar("ASSOCIATION");
my $CN = $idmlib->idmgetvar("CN");
if ($CN eq "bob") {
  # for "bob", check to see if he's been enabled
  my $ENABLE = $idmlib->idmgetvar("REMOVE_Login Disabled");
  if ($ENABLE eq "true") {
    # bob is enabled again
    cmd="appenable -association ". $ASSOCIATION
    $idmlib->exec($cmd);
  }
}
```

Handling Associations

The association value indicates which identity has been changed. If the identity has no association, an association must be generated for it using an implementation-specific rule that you have adopted. When Identity Manager processes an event for an identity with no association, it executes the driver's Matching policy. This policy attempts to match the event's identity to an identity on the external application's system. Doing this usually involves executing a query. The default Matching policy included with the Scripting driver queries for matching Users and Groups based on the CN attribute. If the event's identity matches an identity on the external application, both identities must be assigned the new association. Assigning this association can be done as part of the query-handling script. (Handling queries is discussed in "Handling Query Events" on page 52.) If no identity matches, an Add event is issued, and the new association can be assigned as part of the Add event-handling script:

```
# Adding an association
my $idmlib = new IDMLib();
$idmlib->idmsetvar("COMMAND", "ADD_ASSOCIATION");
$idmlib->idmsetvar("ASSOCIATION", $MyAssociation);
$idmlib->idmsetvar("EVENT_ID", $EVENT_ID);
$idmlib->idmsetvar("DEST_DN", $SRC_DN);
$idmlib->idmsetvar("DEST_ENTRY_ID", $SRC_ENTRY_ID);
```

The above example demonstrates each name/value pair that must be set for an association to be assigned by the Identity Manager engine. The values of EVENT_ID, SRC_DN and SRC_ENTRY_ID are always sent by the engine during an add event, and therefore, are available for your add script to obtain using $idmlib->idmgetvar(). The example above also illustrates the $idmlib->idmsetvar() function. For detailed information on how to use $idmlib->idmsetvar(), see "Perl (IDMLib.pm) Reference" on page 152. This function sets a name and value which indicates what action Identity Manager should perform. For example, the pair COMMAND and ADD_ASSOCIATION instructs the shim to create an add-association document to assign an association to an identity, as discussed above. The pair EVENT_ID and $EVENT_ID instruct the shim to assign add-association document an event-id described by the variable $EVENT_ID. This is important for the engine to match documents sent and returned on the subscriber channel.

The Subscriber can also issue MODIFY_ASSOCIATION and REMOVE_ASSOCIATION commands:

```
# Removing an association
my $idmlib = new IDMLib();
$idmlib->idmsetvar("COMMAND", "REMOVE_ASSOCIATION");
$idmlib->idmsetvar("ASSOCIATION", $MyAssociation);
$idmlib->idmsetvar("EVENT_ID", $EVENT_ID);
$idmlib->idmsetvar("DEST_DN", $SRC_DN);
$idmlib->idmsetvar("DEST_ENTRY_ID", $SRC_ENTRY_ID);

# Modifying an association
my $idmlib = new IDMLib();
$idmlib->idmsetvar("COMMAND", "MODIFY_ASSOCIATION");
$idmlib->idmsetvar("ASSOCIATION", $OldAssociation);
$idmlib->idmsetvar("ASSOCIATION", $NewAssociation);
$idmlib->idmsetvar("EVENT_ID", $EVENT_ID);
$idmlib->idmsetvar("DEST_DN", $SRC_DN);
$idmlib->idmsetvar("DEST_ENTRY_ID", $SRC_ENTRY_ID);
```

Returning an Event Status

On the Subscriber channel, you often do not need Identity Manager to perform an action, but simply need to report a status. The STATUS_ subroutines noted below can be used to indicate a status to Identity Manager. They take a message to be logged as the parameter.

Table 5-8 *STATUS Subroutines*

Subroutine	Identity Manager Action
status_success()	Identity Manager marks the event as a success and submits the next event in the event queue, if any. You should issue this status even if your script does nothing.
status_warning()	The event can be processed, but it might require attention. Identity Manager issues your warning message in its log, and then submits the next event.
status_retry()	The event cannot be processed, but Identity Manager should resubmit the event because it should be able to be processed soon. This status can be issued if your external application appears to be temporarily unavailable. However, this status should be used cautiously because a backlog results if Identity Manager continually retries one event.
status_error()	The event cannot be processed and it should not be resubmitted. Identity Manager issues the error message and submits the next event. You should make a detailed error message so the problem can be corrected.
status_fatal()	For some reason, the driver must be stopped. Identity Manager issues your message and stops the driver. This could be used if the external application appears to be permanently offline. The event remains in the queue and is resubmitted when the driver is restarted.

Examples Using the Status() Functions

```
$idmlib->exec($cmd);
if ($? == 0) {
  $idmlib->status_success("Command was successful");
}
$idmlib->exec($cmd);
if ($? == 0) {
  if ($password eq '') {
    # created, but no password
    $idmlib->status_warning("User added without password");
  }
}
$idmlib->exec($cmd);
if ($? != 0) {
  $idmlib->status_error("Command failed");
}
```

Writing Values

`$idmlib->idmsetvar()` is used to set values to return to Identity Manager. It is passed a name and value. For detailed information on how to use `$idmlib->idmsetvar()`, see "Perl (IDMLib.pm) Reference" on page 152. In the previous `ADD_ASSOCIATION` example, `$idmlib->idmsetvar()` is used to set the `ASSOCIATION` value. You can specify values for items listed in the table above. Generally, `$idmlib->idmsetvar()` is used is to add, modify and delete associations or return information for a query operation. Other information is returned to the shim through other command functions, such as `status_success()`, which use `IDMSETVAR` indirectly.

Handling Query Events

For Query events, Identity Manager submits values that define the parameters of a search of the external application's identity management system. Queries are usually issued from the Policies you have defined for your system. The table below specifies values that can be specified in queries. Not all values are relevant to your external application.

***Table 5-9** Query Values*

Value Name	Description
`SCOPE`	Specifies what identities are searched. A base object is specified with the `ASSOCIATION` or `DEST_DN` values (see below). The value `entry` means that only the base object is searched. The value `subordinates` means that the immediate subordinates of the base object are searched. The value `subtree` (the default) indicates that the base object and all subordinates are searched. The last two values are only relevant in a hierarchical system.
`ASSOCIATION`	The base object for the search. If both `ASSOCIATION` and `DEST_DN` have values, `ASSOCIATION` is used. If neither is specified, the base object is the root of the identity management system.
`DEST_DN`	The base object for the search (see also `ASSOCIATION` above).
`CLASS_NAME`	The base class of the base object.
`EVENT_ID`	An identifier for the event.
`SEARCH_CLASSES`	A list of classes for which to search. Only identities of these classes are returned. If not specified, all identities in the scope matching SEARCH_ATTR_ values are returned (see below)
`SEARCH_ATTRS`	A list of the attribute names specified in SEARCH_ATTR_ values (see below).
`SEARCH_ATTR_`*`attr_name`*	A value that the specified attribute must match. Replace *`attr_name`* with the desired attribute name. Only identities matching all SEARCH_ATTR_ filters are returned.
`READ_ATTRS`	A list of the attribute names whose values are returned for each matching identity.
`ALL_READ_ATTRS`	The presence of this value indicates that all attribute values should be returned for matching identities.

Value Name	Description
NO_READ_ATTRS	The presence of this value indicates that no attribute values should be returned for matching identities.
READ_PARENT	The presence of this value indicates that the parent object of each matching identity should be returned. Only relevant in hierarchical systems.

Execute the query against the external application using application-provided tools. Then return each identity by setting an `INSTANCE` command, followed by relevant values from the list below.

***Table 5-10** Query Values*

Value Name	Description
CLASS_NAME	The class of the identity. Required.
SRC_DN	A distinguished name representing the logical location of the identity in the system (optional).
ASSOCIATION	The association of the identity, if available (optional).
PARENT	The association of the parent object of the identity (optional). Only relevant in hierarchical systems.
ATTR_*attr_name*	A list of values for the attribute specified by *attr_name*. Return attribute values specified by the READ_ATTRS value.

After returning all identities, call `$idmlib->status_success()` to indicate a successful query.

Subscriber Summary and Examples

Below is a more detailed summary of the actions to take for a non-Query event.

1. Gather information about the event using `$idmlib->idmgetvar()`. Return a warning or error if there is a problem.
2. Submit the event data to the external application using application-provided tools.
3. Set event values with `$idmlib->idmsetvar()`.
4. If you have not already done so, set a status with a `$idmlib->status()` subroutine.

Below is an example `add.pl`, which forms an association from an identity's CN and class name, and uses a hypothetical tool called `appadd`.

```
#!/usr/bin/perl

use IDMLib;
my $idmlib = new IDMLib();
my $ClassName = $idmlib->idmgetvar("CLASS_NAME");
my $CN = $idmlib->idmgetvar("CN");
my $PhoneNumber = $idmlib->idmgetvar("Telephone");
my $EVENT_ID = $idmlib->idmgetvar("EVENT_ID");

if (($ClassName eq '') || ($CN eq '')) {
  $idmlib->status_error( "Add event: missing CLASS_NAME and/or CN" );
} else {
  my $Command = "appadd -n $CN -t $PhoneNumber";
  my $rc = $idmlib->exec( $Command );
  if ( $rc == 0 ) {
    $idmlib->idmsetvar("COMMAND", "ADD_ASSOCIATION");
    $idmlib->idmsetvar("ASSOCIATION", $CN . $ClassName);
    $idmlib->idmsetvar("EVENT_ID", $EVENT_ID);
    $idmlib->idmsetvar("DEST_DN", $SRC_DN);
    $idmlib->idmsetvar("DEST_ENTRY_ID", $SRC_ENTRY_ID);

    $idmlib->status_success( "Add event succeeded" );
  } else {
    $idmlib->status_error( "Add event failed with error code" . $rc );
  }
}
```

Handling a query is a similar process, except that you return `INSTANCE` items rather than using other commands. Below is an example `query.pl` that searches an external application for a telephone number. It uses a hypothetical tool called `appsearch`.

```
#!/usr/bin/perl

use IDMLib;
my $idmlib = new IDMLib();
my $SearchName = $idmlib->idmgetvar("SEARCH_ATTR_CN");
my $EVENT_ID = $idmlib->idmgetvar("EVENT_ID");
my $ASSOCIATION = $idmlib->idmgetvar("ASSOCIATION");
my $CLASS_NAME = $idmlib->idmgetvar("CLASS_NAME");

if ($SearchName eq "") {
  $idmlib->status_error( "Query: no search value" );
} else {
  my $Command = "appsearch -n ". $SearchName;
  $Results = `$Command`;
  if ($Results ne "") {
    my @phoneinfo = split(" ", $Results);
    my $Phone = $phoneinfo[0];

    $idmlib->idmsetvar("COMMAND", "INSTANCE");
    $idmlib->idmsetvar("CLASS_NAME", $CLASS_NAME);
    $idmlib->idmsetvar("EVENT_ID", $EVENT_ID);
    $idmlib->idmsetvar("ASSOCIATION", $ASSOCIATION);
    $idmlib->idmsetvar("ATTR_Telephone", $Phone);

    $idmlib->status_success( "Query succeeded" );
  } else {
    # Return success with no results
    $idmlib->status_success( "Query succeeded (no matches)" );
  }
}
```

Publisher Script Development

Events that occur on the external application are submitted to Identity Manager on the Publisher channel. The Scripting driver polls the external application for events periodically. How this poll detects events is implementation-specific and must be defined the user.

Polling for Application Events

The Driver calls `poll.pl` to detect application events. `poll.pl` should be implemented as follows:

1. Use application-provided tools to detect events in your application. (See the discussion in Step Two.)
2. For each event, call the `usclh` changelog tool to submit the event to be published. The changelog tool allows for additional information to be supplied through standard input. This is an appropriate mechanism for passing data that might be too large for command line or too sensitive to appear in a shell's history or environment. For more information on `usclh`, see "Publisher Change Log Tool" on page 175

Below is an example of a `poll.pl` that checks for a password change. It uses a hypothetical application tool called appchg.

```
  #!/usr/bin/perl

  use IDMLib;
  $my idmlib = new IDMLib();
   my $Results = `appchg --passwd-changes`;
  foreach $Result ( split("\n", $Results) ) {
    # Entries are in the format "association:password"
    ($Association, $Password) = split(":", $Result);
    `usclh -t modify-password -a $Association <<EOF
$Password
EOF`;
  }

  # look for attribute values being added to each user
  $Results = `appchg --add-attr-changes`;
  foreach $Result ( split("\n", $Results) ) {
    # Entries are in the format "association:attribute:value"
    ($Association, $Attribute, $Value) = split(":", $Result);
    `usclh -t modify -c User -a $Association <<EOF
ADD_$Attribute=$Value
EOF`;
  }

  # look for attribute values being removed from each user
  $Results = `appchg --remove-attr-changes`;
  foreach $Result ( split("\n", $Results) ) {
    # Entries are in the format "association:attribute:value"
    ($Association, $Attribute, $Value) = split(":", $Result);
    `usclh -t modify -c User -a $Association <<EOF
REMOVE_$Attribute=$Value
EOF`;
  }
```

In the above example, three separate events are submitted to the publisher change log, using the change log tool, `usclh`. The first invocation submits a modify-password event to be published. The second event submits a modify event to be published for an attribute add. The third invocation submits another modify event to be published for an attribute removal. The second and third invocations can be combined into a single modify event, if desired.

Events submitted using `usclh` are processed through your driver's Publisher Channel's policies. See the Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for more information.

Using the Heartbeat Script

Another script executed in the Publisher Channel is `heartbeat.pl`. This script is executed when the Publisher Channel is idle for the interval specified in the Driver parameters. (You can set the interval to 0 so no heartbeat is issued.) You can use the heartbeat to check the availability of the external system or do "idle state" tasks. The `$idmlib->heartbeat_success()`, `$idmlib->heartbeat_warning()`, and `$idmlib->heartbeat_error()` subroutines can be used to indicate the result of the heartbeat. Below is an example based on a hypothetical tool called `apphealth`.

```
my $idmlib = new IDMLib();
my $rc = `apphealth`;
if ($rc == 0) {
  $idmlib->heartbeat_success("Heartbeat succeeded");
} else {
  $idmlib->heartbeat_error("Heartbeat failed with error code " . $rc);
}
```

The response to the heartbeat is implementation-dependent, and can be defined in policies or in the script itself. You could send a message to auditing using NetIQ Audit. You could store a value in a file and have Subscriber scripts read the file and call `$idmlib->heartbeat_retry()` if they find that value in the file.

Other Scripting Topics

- "Driver Parameters" on page 72
- "Querying the Identity Vault" on page 72
- "Tracing and Debugging" on page 73

Driver Parameters

A driver has values known as driver parameters. The driver parameters are divided into driver settings applicable to the whole driver, and Subscriber and Publisher Settings for their respective channels. The `$idmlib->idmgetdrvvar()`, `$idmlib->idmgetsubvar() and $idmlib->idmgetpubvar()` functions can be used to retrieve these values. The table below shows parameters in the default Scripting driver. Other parameters can be added to the driver's XML Configuration file (see the *NetIQ Identity Manager Administration Guide*).

Table 5-11 *Scripting Driver Parameters*

Parameter Name	Driver/Channel	Description	Values
INSTALL_PATH	Driver	The installation path of the Driver	string value
auto-loopback-detection	Driver	Whether to enable automatic loopback detection	true/false
subscriber-script	Subscriber	The root script file for Subscriber events, relative to the driver installation path	string value
pub-polling-interval	Publisher	The interval in seconds between Publisher polls for application events	number
pub-heartbeat-interval	Publisher	The amount of idle time in seconds before a heartbeat event is issued	number
pub-disabled	Publisher	Whether the Publisher Channel (such as for polling) is disabled	true/false

In the following example, a script retrieves the Publisher polling interval.

```
my $PollingInterval = $idmlib->idmgetpubvar("pub-polling-interval");
```

Querying the Identity Vault

Scripts might need to retrieve information from the Identity Vault. They can do this by issuing a query.

1. Execute the query by calling `$idmlib->idmquery($class, $association, $readattrs)` with the appropriate parameters:
   - The first parameter is the class-name
   - The second parameter is the association of the object to query
   - The third parameter are the attributes to read, comma-separated
2. Read the result (instance) using `$idmlib->idmgetqvar()`.

Query support is currently limited. It returns only one instance based on the specified association or DN. If both association and DN are specified, association is used. The functions below allow you to retrieve information from the instance.

The following is an example of a query of the Identity Vault that retrieves the address and ZIP code for user Bob.

```
my $idmlib = new IDMLib();
$idmlib->idmquery("User", "Bob", "SA,Postal Code");

my $Address = $idmlib->idmgetqvar( "SA" );
my $ZIPCode = $idmlib->idmgetqvar( "Postal Code" );
# ... etc. ...
```

Tracing and Debugging

The function IDMTRACE allows you to write a message to the Trace Log. Tracing is useful for script debugging and auditing.

```
  $idmlib->trace("Trace Message");
```

When you develop scripts, you might need to do some debugging to track down problems. The following list indicates some facilities for debugging.

- The Driver traces activity to its Trace file (`logs/trace.log` by default). The trace level setting in `conf/usdrv.conf` controls how much debugging is written to the log.

Trace Level	Description
0	No debugging.
1-3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous level plus Remote Loader, driver, driver shim, and Driver connection messages.
5-7	Previous level plus Change Log and loopback messages. Higher trace levels provide more detail.
8	Previous level plus Driver status log, Driver parameters, Driver command line, Driver security, Driver Web server, Driver schema, Driver encryption, Driver SOAP API, and Driver include/exclude file messages.
9	Previous level plus low-level networking and operating system messages.
10	Previous level plus maximum low-level program details.

The trace level is set using the -trace option in `usdrv.conf`, for example -trace 9.

You can view the trace file through a Web browser:

1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
2. Authenticate by using any username and the password that you specified as the Remote Loader password.
3. Click **Trace**.

- The eDirectory tool DSTrace can be used to monitor Identity Manager activity. Set the tracing level for the Driver in Identity Console. DSTrace shows the XML documents being submitted to the driver for events and how policies are evaluated. It also shows the status and message for each event.
- The Status Log is written to `logs/dirxml.log`. It shows a summary of the events that have been recorded on the Subscriber and Publisher channels.

  You can view the Status Log through a Web browser:

  1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
  2. Authenticate by using any username and the password that you specified as the Remote Loader password.
  3. Click **Status**.

Although it is best to start the driver in production environments from the startup script, you can run usdrv manually. When you do so, any text written to standard output from scripts is displayed in the interactive shell.

Deployment

The Scripting driver is installed by using a setup program. See "Installing the Linux and UNIX Scripting Driver Shim" on page 19, for more information on installing the default driver.

Deploying a Custom Driver

To deploy your custom driver, the end user should first run the Scripting driver installation program provided by the installation media (see"Installing the Linux and UNIX Scripting Driver Shim" on page 19). This program installs core files needed by the driver. Then, your custom driver files can be deployed in any convenient way, whether through an installation program or even simply an archive file. The table below shows the directory structure below the installation directory and what files are installed.

Table 5-12 *Directory Structure and Files*

Directory	Description	Required Files

`bin/`	Location of executable programs	`usdrv`
		`ussmh`
		`usclh`

`changelog/`	Used for Publisher event processing	None
`conf/`	Location of the driver shim configuration file	`usdrv.conf` (customized)
`keys/`	Location of security key files	None
`logs/`	Location of log files	None

Directory	Description	Required Files
`loopback/`	Used for automatic loopback detection	None
`rules/`	Location of Driver configuration file	`Scripting.xml` (customized)
`schema/`	Location of schema files	`schema.def` (customized)
`scripts/`	Location of script files	Those required by your Driver (customized)

On Linux and UNIX, the Scripting driver is installed to `/opt/novell/usdrv`.

The formats of `usdrv.conf` and `schema.def` can be viewed in "The Driver Shim Configuration File" on page 32 and "The Connected System Schema File" on page 38.

If SSL encryption is desired for communication between the driver shim and Identity Manager engine, a certificate must be retrieved from the Identity Vault. Run `usdrv -s` and follow the prompts to retrieve the certificate, which will be stored in the `keys/` directory. You must have LDAP with SSL available for the Metadirectory. When making an installation program for deployment, you might want to run `usdrv -s` as part of the installation.

To ensure that only authorized systems access the Metadirectory, a Driver object password and Remote Loader password are used. Run `usdrv -sp` and enter the passwords at the prompts. This action can be incorporated into an installation program.

You should distribute the XML configuration file that contains parameters and policies your Driver needs. The user can then select it when installing your Driver.

Python Developer Guide

The Scripting driver provides a complete Python API for interacting with identity management systems whose tools (including APIs) are available on Linux and UNIX. The Identity Vault and Identity Manager can run on any supported operating system. Identity Manager can communicate with any supported system on which the driver is installed via an encrypted network connection.

Before beginning script development, review the preceding topics in this section for information on defining what data is synchronized between identity management systems.

With additional development work, the driver can also be customized to support any scripting language that supports command-line operation.

Developing a custom driver with Python scripts is discussed in this section. Topics include

- "Application Tools Evaluation" on page 76
- "Policy and Script Development" on page 77
- "Deployment" on page 88

Application Tools Evaluation

To change the data in your external application, you need to know how to use the application's tools or API (Application Programming Interface). These tools must provide automated operation and not require user interaction.

Application Command Line Tools

An application often provides command line tools. These tools are manually executed from the command line, and they can be executed from scripts. For example, suppose the application provides a tool to add identities with a program called appadd.

```
appadd -n "Bob Smith" -t "818-555-2100"
```

This command adds an identity named "Bob Smith" with the specified phone number. The strings following the program name are called parameters or arguments. The Linux and UNIX Scripting driver provides a function called exec to execute external programs, log the command to the system log, and produce a status document indicating the level of success.

```
from idmlib import *
CommandLine = "appadd -n $UserName -t $PhoneNumber"
exec(CommandLine($CommandLine)
```

For command line tools, you can construct the command line's parameters using the values passed to the script, then execute the program.

Application Event Monitoring

You also need to determine what tools are available for monitoring event changes in the application. The Scripting driver works on a polling system. It periodically calls a polling script to determine what has changed in the external application. You can use the following ideas for monitoring changes:

- The first time the polling script is run, a list of identities and relevant attributes is read from the application using an application-provided tool. This list is stored as a file. On subsequent polls, a new list is generated and compared to the old list. Any differences are submitted as events to the driver.
- The application provides a tool that allows you to request all identities that have changed after a certain point in time. The polling script requests events that have occurred since the previous poll.
- The application allows a script to be run when an event occurs. You can write a script that stores the event data into a file. When the Script driver polling script runs, it consumes this file and submits the data as an event to the driver using the `usclh` change log tool. For detailed information on `usclh`, see "Publisher Change Log Tool" on page 175.

Monitoring the application's changes might be the most difficult aspect of developing your driver. You must study your application's tools to determine the best way to achieve synchronization.

Policy and Script Development

At this point you should have a list of what data will be synchronized, how events will be handled, and what application tools are available. It is time to develop the heart of your driver in policies and scripts.

Many types of tasks can be handled in driver policies. You can import the driver configuration provided with the Scripting driver, and then edit policies in NetIQ Identity Console. You can also edit policies and simulate their operation in NetIQ Designer. The extensive functionality of policies is outside the scope of this document, and so you should refer to your Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for help.

It's often difficult to write complex tasks inside policies, such as executing external commands, processing input and output, and file I/O. Tasks requiring such operations are better suited in scripts, where an entire language environment and tools are available. You can also accomplish many of the operations performed in policies, so if you are more familiar with your scripting language than policies, you can develop your driver more quickly by using scripts. Scripting languages such as Perl, Python and Shell scripts offer an environment that is often well suited for your target application's APIs or developer kits. For example, your target application might already contain Perl of Python library routines for manipulating the application's identities.

Event Data Format

Event data is submitted to the scripts in name/value pair format. This format consists of lines containing a name, an equal sign (=) and a value. Therefore, each line is a name/value pair. Each name/value pair is unique, but there can be multiple name/value pairs with identical names but different values.

```
ASSOCIATION=BobUser
ADD_TELEPHONE=818-555-2100
ADD_TELEPHONE=818-555-9842
```

You typically don't need to worry about the format. The script library provides functions for retrieving event data independent of its format.

Subscriber Script Development

After all Policy processing is complete, Identity Manager submits the event in XML format to the driver shim. The driver shim submits the event data to the scripts.

In the default Scripting driver, the `subscriber.py` script in the `scripts` folder is called. This script does some preliminary processing, and then calls a routine from an included script. The included scripts correspond to the Subscriber event types: `add.py`, `modify.py`, `modify-password.py`, `delete.py`, `rename.py`, `move.py`, and `query.py`.

For each event type, you should retrieve the information you need from the event data, submit changes to the external application using application-provided tools and return a status (such as success or failure) to Identity Manager.

Event data is retrieved primarily by using the `idmgetvar()` function. This function returns an array of values corresponding to the name specified as the function's parameter. The following table shows many item names.

Table 5-13 Item Names

Name	Description
COMMAND	The command for the event, usually indicating the event type. Possible values are: `add`, `modify`, `delete`, `rename`, `modify-password`, `check-object-password`.
ASSOCIATION	The identifier that distinguishes an identity on both identity management systems.
CLASS_NAME	An identity's class, such as User or Group.
SRC_DN	An identity's distinguished name (DN) in the namespace of the source (sender), in slash format.
EVENT_ID	An identifier for the event, for internal use.
SRC_ENTRY_ID	An identifier for the identity that generated the event, in the namespace of the source (sender).
DEST_DN	An identity's distinguished name (DN) in the namespace of the destination (receiver), in slash format.
DEST_ENTRY_ID	An identifier for an entry in the namespace of the destination (receiver).
ADD_*attr_name*	A value to be added to an identity, for attribute *attr_name*.
REMOVE_*attr_name*	A value to be removed from an identity, for attribute *attr_name*.
ADD_REF_*attr_name*	A value to be added to attribute *attr_name*, where the value is an association referring to another identity.
REMOVE_REF_*attr_name*	A value to be removed from attribute *attr_name*, where the value is an association referring to another identity.
OLD_PASSWORD	The previous password for an identity that has changed its password. Used in Modify Password events.
PASSWORD	The new password for an identity. Used in Add and Modify Password events.
OLD_SRC_DN	The distinguished name of an identity before a Move or Rename event.
REMOVE_OLD_NAME	Specifies whether an old relative distinguished name should be deleted or retained. Used in Rename events.
STATUS_LEVEL	The status of an event: success, warning, retry, error or fatal.
STATUS_MESSAGE	A message to report with a status.
STATUS_TYPE	A type of status, such as heartbeat.

Examples Of Obtaining Event Data

Example 1:

```
from idmlib import *

command = idmgetvar("COMMAND")
# check for an add event
if command = "add":
  # call the add script
  os.system("add.py")
```

Example 2:

```
from idmlib import *

# obtain the event's association and CN attribute
association = idmgetvar("ASSOCIATION")
CN = idmgetvar("CN")
if CN = "bob":
  # for "bob", check to see if he's been enabled
  ENABLE = idmgetvar("REMOVE_Login Disabled")
  if ENABLE = "true":
    # bob is enabled again
    cmd = "appenable -association " + ASSOCIATION
    exec(cmd)
```

Handling Associations

The association value indicates which identity has been changed. If the identity has no association, an association must be generated for it using an implementation-specific rule that you have adopted. When Identity Manager processes an event for an identity with no association, it executes the driver's Matching policy. This policy attempts to match the event's identity to an identity on the external application's system. Doing this usually involves executing a query. The default Matching policy included with the Scripting driver queries for matching Users and Groups based on the CN attribute. If the event's identity matches an identity on the external application, both identities must be assigned the new association. Assigning this association can be done as part of the query-handling script. (Handling queries is discussed in "Handling Query Events" on page 52.) If no identity matches, an Add event is issued, and the new association can be assigned as part of the Add event-handling script:

```
# Adding an association
from idmlib import *

idmsetvar("COMMAND", "ADD_ASSOCIATION")
idmsetvar("ASSOCIATION", MyAssociation)
idmsetvar("EVENT_ID", EVENT_ID)
idmsetvar("DEST_DN", SRC_DN)
idmsetvar("DEST_ENTRY_ID", SRC_ENTRY_ID)
```

The above example demonstrates each name/value pair that must be set for an association to be assigned by the Identity Manager engine. The values of EVENT_ID, SRC_DN and SRC_ENTRY_ID are always sent by the engine during an add event, and therefore, are available for your add script to obtain using `idmgetvar()`. The example above also illustrates the `idmsetvar()` function. For detailed information on how to use `idmsetvar()`, see "Python (idmlib.py) Reference" on page 155. This function sets a name and value which indicates what action Identity Manager should perform. For example, the pair "COMMAND" and "ADD_ASSOCIATION" instructs the shim to create an add-association document to assign an association to an identity, as discussed above. The pair

"EVENT_ID" and EVENT_ID instruct the shim to assign add-association document an event-id described by the variable EVENT_ID. This is important for the engine to match documents sent and returned on the subscriber channel.

The Subscriber can also issue MODIFY_ASSOCIATION and REMOVE_ASSOCIATION commands:

```
# Removing an association
from idmlib import *

idmsetvar("COMMAND", "REMOVE_ASSOCIATION")
idmsetvar("ASSOCIATION", MyAssociation)
idmsetvar("EVENT_ID", EVENT_ID)
idmsetvar("DEST_DN", SRC_DN)
idmsetvar("DEST_ENTRY_ID", SRC_ENTRY_ID)

# Modifying an association
my idmlib import *

idmsetvar("COMMAND", "MODIFY_ASSOCIATION")
idmsetvar("ASSOCIATION", OldAssociation)
idmsetvar("ASSOCIATION", NewAssociation)
idmsetvar("EVENT_ID", EVENT_ID)
idmsetvar("DEST_DN", SRC_DN)
idmsetvar("DEST_ENTRY_ID", SRC_ENTRY_ID)
```

Returning an Event Status

On the Subscriber channel, you often do not need Identity Manager to perform an action, but simply need to report a status. The status_ subroutines noted below can be used to indicate a status to Identity Manager. They take a message to be logged as the parameter.

Table 5-14 *STATUS Subroutines*

Subroutine	Identity Manager Action
status_success()	Identity Manager marks the event as a success and submits the next event in the event queue, if any. You should issue this status even if your script does nothing.
status_warning()	The event can be processed, but it might require attention. Identity Manager issues your warning message in its log, and then submits the next event.
status_retry()	The event cannot be processed, but Identity Manager should resubmit the event because it should be able to be processed soon. This status can be issued if your external application appears to be temporarily unavailable. However, this status should be used cautiously because a backlog results if Identity Manager continually retries one event.
status_error()	The event cannot be processed and it should not be resubmitted. Identity Manager issues the error message and submits the next event. You should make a detailed error message so the problem can be corrected.

Subroutine	Identity Manager Action
status_fatal()	For some reason, the driver must be stopped. Identity Manager issues your message and stops the driver. This could be used if the external application appears to be permanently offline. The event remains in the queue and is resubmitted when the driver is restarted.

Examples Using the Status() Functions

```
rc = exec(cmd)
if (rc = 0):
  status_success("Command was successful")
rc = exec(cmd)
if (rc == 0):
  if (password eq "")
    # created, but no password
    status_warning("User added without password")

rc = exec(cmd)
if (rc != 0):
  status_error("Command failed")
```

Writing Values

The `idmsetvar()` function is used to set values to return to Identity Manager. It is passed a name and value. For detailed information on how to use `idmsetvar()`, see "Python (idmlib.py) Reference" on page 155. In the previous ADD_ASSOCIATION example, `idmsetvar()` is used to set the ASSOCIATION value. You can specify values for items listed in the table above. Generally, `idmsetvar()` is used is to add, modify and delete associations or return information for a query operation. Other information is returned to the shim through other command functions, such as `status_success()`, which use `idmsetvar()` indirectly.

Handling Query Events

For Query events, Identity Manager submits values that define the parameters of a search of the external application's identity management system. Queries are usually issued from the Policies you have defined for your system. The table below specifies values that can be specified in queries. Not all values are relevant to your external application.

Table 5-15 *Query Values*

Value Name	Description
SCOPE	Specifies what identities are searched. A base object is specified with the ASSOCIATION or DEST_DN values (see below). The value `entry` means that only the base object is searched. The value `subordinates` means that the immediate subordinates of the base object are searched. The value `subtree` (the default) indicates that the base object and all subordinates are searched. The last two values are only relevant in a hierarchical system.

Value Name	Description
ASSOCIATION	The base object for the search. If both ASSOCIATION and DEST_DN have values, ASSOCIATION is used. If neither is specified, the base object is the root of the identity management system.
DEST_DN	The base object for the search (see also ASSOCIATION above).
CLASS_NAME	The base class of the base object.
EVENT_ID	An identifier for the event.
SEARCH_CLASSES	A list of classes for which to search. Only identities of these classes are returned. If not specified, all identities in the scope matching SEARCH_ATTR_ values are returned (see below)
SEARCH_ATTRS	A list of the attribute names specified in SEARCH_ATTR_ values (see below).
SEARCH_ATTR_*attr_name*	A value that the specified attribute must match. Replace *attr_name* with the desired attribute name. Only identities matching all SEARCH_ATTR_ filters are returned.
READ_ATTRS	A list of the attribute names whose values are returned for each matching identity.
ALL_READ_ATTRS	The presence of this value indicates that all attribute values should be returned for matching identities.
NO_READ_ATTRS	The presence of this value indicates that no attribute values should be returned for matching identities.
READ_PARENT	The presence of this value indicates that the parent object of each matching identity should be returned. Only relevant in hierarchical systems.

When a query invokes your query script, use the parameter information to query your external application using application-provided tools. Then return each identity by setting an INSTANCE command, followed by relevant values from the list below.

Table 5-16 *Query Values*

Value Name	Description
CLASS_NAME	The class of the identity. Required.
SRC_DN	A distinguished name representing the logical location of the identity in the system (optional).
ASSOCIATION	The association of the identity, if available (optional).
PARENT	The association of the parent object of the identity (optional). Only relevant in hierarchical systems.
ATTR_*attr_name*	A list of values for the attribute specified by *attr_name*. Return attribute values specified by the READ_ATTRS value.

After returning all identities, call status_success() to indicate a successful query.

Subscriber Summary and Examples

Below is a more detailed summary of the actions to take for a non-Query event.

1 Gather information about the event using `idmgetvar()`. Return a warning or error if there is a problem.

2 Submit the event data to the external application using application-provided tools.

3 Set event values with `idmsetvar()`.

4 If you have not already done so, set a status with a `status()` subroutine.

Below is an example `add.py`, which forms an association from an identity's CN and class name, and uses a hypothetical tool called appadd.

```
#!/usr/bin/python

from idmlib import *

ClassName = idmgetvar("CLASS_NAME")
CN = idmgetvar("CN")
PhoneNumber = idmgetvar("Telephone")
EventId = idmgetvar("EVENT_ID")
SrcDn = idmgetvar("SRC_DN")
SrcEntryId = idmgetvar("SRC_ENTRY_ID")
if ClassName = "" || CN = "":
  status_error("Add event: missing CLASS_NAME and/or CN")
else:
  Command = "appadd -n " + CN + -t " + PhoneNumber
  rc = exec(Command)
  if rc = 0:
    idmsetvar("COMMAND", "ADD_ASSOCIATION")
    idmsetvar("ASSOCIATION", CN + ClassName)
    idmsetvar("EVENT_ID", EventId)
    idmsetvar("DEST_DN", SrcDn)
    idmsetvar("DEST_ENTRY_ID", SrcEntryId)
    status_success("Add event succeeded")
  else:
    status_error("Add event failed with error code" + rc)
```

Handling a query is a similar process, except that you return `INSTANCE` items rather than using other commands. Below is an example `query.py` that searches an external application for a telephone number. It uses a hypothetical tool called `appsearch`.

```
#!/usr/bin/python

import os
from idmlib import *

SearchName = idmgetvar("SEARCH_ATTR_CN")
EventId = idmgetvar("EVENT_ID")
Association = idmgetvar("ASSOCIATION")
ClassName = idmgetvar("CLASS_NAME")
if SearchName = "":
  status_error("Query: no search value")
else:
  Command = "appsearch -n " + SearchName
  Results = os.popen(Command, ‘r').readlines()
  if Results != "":
    phoneinfo = split(" ", Results)
    Phone = phoneinfo[0]
    idmsetvar("COMMAND", "INSTANCE")
    idmsetvar("CLASS_NAME", ClassName)
    idmsetvar("EVENT_ID", EventId)
    idmsetvar("ASSOCIATION", Association)
    idmsetvar("ATTR_Telephone", Phone)
    status_success("Query succeeded")
  else:
    # Return success with no results
    status_success("Query succeeded (no matches)")
```

Publisher Script Development

Events that occur on the external application are submitted to Identity Manager on the Publisher channel. The Scripting driver polls the external application for events periodically. How this poll detects events is implementation-specific and must be defined by the user.

Polling for Application Events

The Driver calls `poll.py` to detect application events. `poll.py` should be implemented as follows:

1. Use application-provided tools to detect events in your application (see discussion in Step 2).
2. For each event, call the usclh changelog tool to submit the event to be published. The changelog tool allows for additional information to be supplied through standard input. This is an appropriate mechanism for passing data that might be too large for command line or too sensitive to appear in a shell’s history or environment. For more information on `usclh`, see “Publisher Change Log Tool” on page 175

Below is an example of a `poll.py` that checks for a password change. It uses a hypothetical application tool called appchg.

```
#!/usr/bin/python

import os
from idmlib import *

# look for password changes with our ficticious app, appchg
results = os.popen("appchg --passwd-changes", 'r').readlines()
for result in results:
  fields = result.split(":")
  Association = fields[0]
  Password = fields[1]
  # submit event to the publisher changelog
  usclh = os.popen(usclh -t modify-password -a " + Association, 'w')
  usclh.write(Password)
  usclh.close()

# look for attribute values being added to each user
results = os.popen("appchg --add-attr-changes")
for result in results:
  # Entries are in the format "association:attribute:value"
  fields = result.split(":")
  Association = fields[0]
  Attribute = fields[1]
  Value = fields[2]
  # submit event to the publisher changelog
  local_usclh = os.environ.get('INSTALL_PATH') + "bin/usclh"
  usclh = os.popen(local_usclh + " -t modify -c User -a " + Association,
'w')
  usclh.write("ADD_" + Attribute + "=" + Value)
  usclh.close()

# look for attribute values being removed from each user
results = os.popen("appchg --add-attr-changes")
for result in results:
  # Entries are in the format "association:attribute:value"
  fields = result.split(":")
  Association = fields[0]
  Attribute = fields[1]
  Value = fields[2]
  # submit event to the publisher changelog
  local_usclh = os.environ.get('INSTALL_PATH') + "bin/usclh"
  usclh = os.popen(local_usclh + " -t modify -c User -a " + Association,
'w')
  usclh.write("REMOVE_" + Attribute + "=" + Value)
  usclh.close()
```

In the above example, three separate events are submitted to the publisher change log, using the changelog tool, `usclh`. The first invocation submits a `modify-password` event to be published. The second event submits a modify event to be published for an attribute add. The third invocation submits another modify event to be published for an attribute removal. The second and third invocations can be combined into a single modify event, if desired.

Events submitted using `usclh` are processed through your driver's Publisher Channel's policies. See the Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for more information.

Using the Heartbeat Script

Another script executed in the Publisher Channel is `heartbeat.py`. This script is executed when the Publisher Channel is idle for the interval specified in the Driver parameters. (You can set the interval to 0 so no heartbeat is issued.) You can use the heartbeat to check the availability of the external system or do "idle state" tasks. The `heartbeat_success()`, `heartbeat_warning()`, and `heartbeat_error()` subroutines can be used to indicate the result of the heartbeat. Below is an example based on a hypothetical tool called `apphealth`.

```
from idmlib import *

rc = os.sytem("apphealth")
if (rc = 0):
  heartbeat_success("Heartbeat succeeded")
else:
  heartbeat_error("Heartbeat failed with error code " + rc)
```

Other Scripting Topics

- "Driver Parameters" on page 86
- "Querying the Identity Vault" on page 87
- "Tracing and Debugging" on page 87

Driver Parameters

A driver has values known as driver parameters. The driver parameters are divided into driver settings applicable to the whole driver, and Subscriber and Publisher Settings for their respective channels. The `idmgetdrvvar()`, `idmgetsubvar()` and `idmgetpubvar()` functions can be used to retrieve these values. The table below shows parameters in the default Scripting driver. Other parameters can be added to the driver's XML Configuration file (see the *NetIQ Identity Manager 3.6.1 Administration Guide*).

Table 5-17 *Scripting Driver Parameters*

Parameter Name	Driver/Channel	Description	Values
INSTALL_PATH	Driver	The installation path of the Driver	string value
auto-loopback-detection	Driver	Whether to enable automatic loopback detection	true/false
subscriber-script	Subscriber	The root script file for Subscriber events, relative to the driver installation path	string value
pub-polling-interval	Publisher	The interval in seconds between Publisher polls for application events	number

Parameter Name	Driver/Channel	Description	Values
pub-heartbeat-interval	Publisher	The amount of idle time in seconds before a heartbeat event is issued	number
pub-disabled	Publisher	Whether the Publisher Channel (such as for polling) is disabled	true/false

In the following example, a script retrieves the Publisher polling interval:

```
PollingInterval = idmgetpubvar("pub-polling-interval")
```

Querying the Identity Vault

Scripts might need to retrieve information from the Identity Vault. They can do this by issuing a query.

1. Execute the query by calling `idmquery(class, association, readattrs)` with the appropriate parameters:
   - The first parameter is the class-name
   - The second parameter is the association of the object to query
   - The third parameter are the attributes to read, comma-separated
2. Read the result (instance) using `idmgetqvar()`.

Query support is currently limited. It returns only one instance based on the specified association or DN. If both association and DN are specified, association is used. The functions below allow you to retrieve information from the instance.

The following is an example of a query of the Identity Vault that retrieves the address and ZIP code for user Bob.

```
from idmlib import *

idmquery("User", "Bob", "SA,Postal Code")
Address = idmgetqvar("SA")
ZIPCode = idmgetqvar("Postal Code")
# ... etc. ...
```

Tracing and Debugging

The function `trace()` allows you to write a message to the Trace Log. Tracing is useful for script debugging and auditing.

```
from idmlib import *

trace("Trace Message")
```

When you develop scripts, you might need to do some debugging to track down problems. The following list indicates some facilities for debugging.

- The Driver traces activity to its Trace file (`logs/trace.log` by default). The trace level setting in `conf/usdrv.conf` controls how much debugging is written to the log.

Trace Level	Description
0	No debugging.
1-3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous level plus Remote Loader, driver, driver shim, and Driver connection messages.
5-7	Previous level plus Change Log and loopback messages. Higher trace levels provide more detail.
8	Previous level plus Driver status log, Driver parameters, Driver command line, Driver security, Driver Web server, Driver schema, Driver encryption, Driver SOAP API, and Driver include/exclude file messages.
9	Previous level plus low-level networking and operating system messages.
10	Previous level plus maximum low-level program details.

The trace level is set using the -trace option in `usdrv.conf`, for example -trace 9.

You can view the trace file through a Web browser:

1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
2. Authenticate by using any username and the password that you specified as the Remote Loader password.
3. Click **Trace**.

- The eDirectory tool DSTrace can be used to monitor Identity Manager activity. Set the tracing level for the Driver in Identity Console. DSTrace shows the XML documents being submitted to the driver for events and how policies are evaluated. It also shows the status and message for each event.
- The Status Log is written to `logs/dirxml.log`. It shows a summary of the events that have been recorded on the Subscriber and Publisher channels.

  You can view the Status Log through a Web browser:

  1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
  2. Authenticate by using any username and the password that you specified as the Remote Loader password.
  3. Click **Status**.

Although it is best to start the driver in production environments from the startup script, you can run usdrv manually. When you do so, any text written to standard output from scripts is displayed in the interactive shell.

Deployment

The Scripting driver is installed by using a setup program. See "Installing the Linux and UNIX Scripting Driver Shim" on page 19, for more information on installing the default driver.

Deploying a Custom Driver

To deploy your custom driver, the end user should first run the Scripting driver installation program provided by the installation media (see“Installing the Linux and UNIX Scripting Driver Shim” on page 19). This program installs core files needed by the driver. Then, your custom driver files can be deployed in any convenient way, whether through an installation program or even simply an archive file. The table below shows the directory structure below the installation directory and what files are installed.

***Table 5-18** Directory Structure and Files*

Directory	Description	Required Files

`bin/`	Location of executable programs	`usdrv`
		`ussmh`
		`usclh`

`changelog/`	Used for Publisher event processing	None
`conf/`	Location of the driver shim configuration file	`usdrv.conf` (customized)
`keys/`	Location of security key files	None
`logs/`	Location of log files	None
`loopback/`	Used for automatic loopback detection	None
`rules/`	Location of Driver configuration file	`Scripting.xml` (customized)
`schema/`	Location of schema files	`schema.def` (customized)
`scripts/`	Location of script files	Those required by your Driver (customized)

On Linux and UNIX, the Scripting driver is installed to `/opt/novell/usdrv`.

The formats of `usdrv.conf` and `schema.def` can be viewed in “The Driver Shim Configuration File” on page 32 and “The Connected System Schema File” on page 38.

If SSL encryption is desired for communication between the driver shim and Identity Manager engine, a certificate must be retrieved from the Identity Vault. Run `usdrv -s` and follow the prompts to retrieve the certificate, which will be stored in the `keys/` directory. You must have LDAP with SSL available for the Metadirectory. When making an installation program for deployment, you might want to run `usdrv -s` as part of the installation.

To ensure that only authorized systems access the Metadirectory, a Driver object password and Remote Loader password are used. Run `usdrv -sp` and enter the passwords at the prompts. This action can be incorporated into an installation program.

You should distribute the XML configuration file that contains parameters and policies your Driver needs. The user can then select it when installing your Driver.

Microsoft VBScript Developer Guide

The Scripting driver provides a complete Microsoft VBScript API for interacting with identity management systems whose tools (including APIs) are available on Windows. The Identity Vault and Identity Manager can run on any supported operating system. Identity Manager can communicate with any supported system on which the driver is installed via an encrypted network connection.

Before beginning script development, review the preceding topics in this section for information on defining what data is synchronized between identity management systems.

With additional development work, the driver can also be adapted to support any scripting language that supports command line operation.

Developing a custom driver with VBScript is discussed in this section. Topics include

Application Tools Evaluation

To change the data in your external application, you need to know how to use the application's tools or API (Application Programming Interface). These tools must provide automated operation and not require user input.

Application Command Line Tools

An application often provides command line tools. These tools are manually executed from the Windows command prompt, and they can be executed from scripts. For example, suppose the application provides a tool to add identities with a program called `appadd.exe`.

```
appadd -n "Bob Smith" -t "818-555-2100"
```

This command adds an identity named "Bob Smith" with the specified phone number. The strings following the program name are called parameters or arguments. The Scripting driver provides a function called `IDMExecute` to execute external programs.

```
CommandLine = "appadd -n " & UserName & " -t " & PhoneNumber
ExitCode = IDMExecute(CommandLine)
```

There is also a function called `IDMExecuteIO` that allows you to pass information on the program's standard input, and receive output from the program's standard output and standard error.

```
Dim Input(1)

Input(0) = "USERNAME=bobsmith"
Input(1) = "TELEPHONE=818-555-2100"

Output = IDMExecuteIO("appadd", Input)

ExitCode = Output(0)
```

`IDMExecuteIO'`s first parameter is the command line, and its second parameter is an array of strings (or Empty) that is submitted as lines to the command program. It returns an array, the first element of which is the program's exit code, and then strings that represent lines returned on the program's standard output and standard error.

For command line tools, you can construct the command line's parameters using the values passed to the script, then execute the program.

Application Objects

Another way to modify application data is through Windows COM (Common Object Model) objects. Consult your application's documentation to see whether it exposes any COM objects. These COM objects can be loaded directly in VBScript:

```
Set AppObject = WScript.CreateObject("MyApplication.MyObject")
AppObject.AddIdentity("Bob Smith", "818-555-2100")
```

There are no guarantees regarding what types of tools are available, or even whether any tools are available. You must determine if sufficient tools are provided by the application. If they are not, you can contact the application's developers and request that such tools be made available.

You should make a list of what tools can be used for each event type. The application might provide one program that can be used for any event type, or it might provide multiple tools.

Application Event Monitoring

You also need to determine what tools are available for monitoring event changes in the application. The Scripting driver works on a polling system. It periodically calls a polling script to determine what has changed in the external application. You can use the following ideas for monitoring changes:

- The first time the polling script is run, a list of identities and relevant attributes is read from the application using an application-provided tool. This list is stored as a file. On subsequent polls, a new list is generated and compared to the old list. Any differences are submitted as events to the driver.
- The application provides a tool that allows you to request all identities that have changed after a certain point in time. The polling script requests events that have occurred since the previous poll.
- The application allows a script to be run when an event occurs. You write a script that stores the event data into a file. When the Scripting driver polling script runs, it consumes this file and submits the data as an event to the driver.

Monitoring the application's changes might be the most difficult aspect of developing your driver. You must study your application's tools to determine the best way to achieve synchronization.

Policy and Script Development

At this point you should have a list of what data will be synchronized, how events will be handled, and what application tools are available. It is time to develop the heart of your driver in policies and scripts.

Many types of tasks can be handled in driver policies. You can import the driver configuration provided with the Scripting driver, and then edit policies in NetIQ Identity Console. You can also edit policies and simulate their operation in NetIQ Designer. The extensive functionality of policies is outside the scope of this document, so you should refer to your Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for help.

Tasks that don't interact with the external application might be more suited to policies. On the other hand, if you are more familiar with your scripting language than policies, you can develop your driver more quickly by using scripts.

Event Data Format

Event data is submitted to the scripts in name/value pair format. This format consists of lines containing a name, an equal sign (=) and a value. Therefore, each line is a name/value pair. Each name/value pair is unique, but there can be multiple name/value pairs with identical names but different values.

```
ASSOCIATION=BobUser
ADD_TELEPHONE=818-555-2100
ADD_TELEPHONE=818-555-9842
```

You typically don't need to worry about the format. The script library provides functions for retrieving event data.

Subscriber Script Development

After all policy processing is complete, Identity Manager submits the event in XML format to the driver shim. The driver shim submits the event data to the scripts.

In the default Scripting driver, the `Subscriber.wsf script` in the `scripts` folder is called. This script does some preliminary processing, and then calls a routine from an included script. The included scripts correspond to the Subscriber event types: `Add.vbs`, `Modify.vbs`, `ModifyPassword.vbs`, `Delete.vbs`, `Rename.vbs`, `Move.vbs`, and `Query.vbs`.

For each event type, you should retrieve the information you need from the event data, submit changes to the external application using application-provided tools and return a status (such as success or failure) to Identity Manager.

Event data is retrieved primarily using the `IDMGetEventValues` function. This function returns an array of values corresponding to the name specified as the function's parameter. (`IDMGetEventValue` is available for single-valued items.) The following table shows many item names.

Table 5-19 Item Names

Value	Description
COMMAND	The command for the event, usually indicating the event type.
ASSOCIATION	The identifier that distinguishes an identity on both identity management systems.
CLASS_NAME	An identity's class, such as User or Group.
SRC_DN	An identity's distinguished name (DN) in the namespace of the source (sender), in slash format.
EVENT_ID	An identifier for the event, for internal use.
SRC_ENTRY_ID	An identifier for the identity that generated the event, in the namespace of the source (sender).
DEST_DN	An identity's distinguished name (DN) in the namespace of the destination (receiver), in slash format.
DEST_ENTRY_ID	An identifier for an entry in the namespace of the destination (receiver).
ADD_*attr_name*	A value to be added to an identity, for attribute *attr_name*.
REMOVE_*attr_name*	A value to be removed from an identity, for attribute *attr_name*.
ADD_REF_*attr_name*	A value to be added to attribute *attr_name*, where the value is an association referring to another identity.
REMOVE_REF_*attr_name*	A value to be removed from attribute *attr_name*, where the value is an association referring to another identity.
OLD_PASSWORD	The previous password for an identity that has changed its password. Used in Modify Password events.
PASSWORD	The new password for an identity. Used in Add and Modify Password events.
OLD_SRC_DN	The distinguished name of an identity before a Move or Rename event.
REMOVE_OLD_NAME	Specifies whether an old relative distinguished name should be deleted or retained. Used in Rename events.
STATUS_LEVEL	The status of an event: success, warning, retry, error or fatal.
STATUS_MESSAGE	A message to report with a status.
STATUS_TYPE	A type of status, such as heartbeat.

Handling Associations

The association value indicates which identity has been changed. If the identity has no association, an association must be generated for it using an implementation-specific rule that you have adopted. When Identity Manager processes an event for an identity with no association, it executes the driver's Matching policy. This policy attempts to match the event's identity to an identity on the external application's system. Doing this usually involves executing a query. The default Matching policy included with the Scripting driver queries for matching Users and Groups based on the CN attribute. If the event's identity matches an identity on the external application, both identities must be assigned the new association. Assigning this association can be done as part of the query-handling script. (Handling queries is discussed in "Handling Query Events" on page 52.) If no identity matches, an Add event is issued, and the new association can be assigned as part of the Add event-handling script:

```
IDMSetCommand "ADD_ASSOCIATION"
IDMWriteValue "ASSOCIATION", MyAssociation
          (etc.)
```

The example above also illustrates the `IDMSetCommand` function. This function sets a command value which indicates what action Identity Manager should perform. The `ADD_ASSOCIATION` command assigns an association to an identity, as discussed above. The Subscriber can also issue `MODIFY_ASSOCIATION` and `REMOVE_ASSOCIATION` commands.

Returning an Event Status

On the Subscriber channel, you often do not need Identity Manager to perform an action, but simply need to report a status. The `IDMStatus` subroutines noted below can be used to indicate a status to Identity Manager. They take a message to be logged as the parameter.

Table 5-20 *Subroutines*

Subroutine	Identity Manager Action
IDMStatusSuccess	Identity Manager marks the event as a success and submits the next event in the event queue, if any. You should issue this status even if your script does nothing.
IDMStatusWarning	The event can be processed, but it might require attention. Identity Manager issues your warning message in its log, and then submits the next event.
IDMStatusRetry	The event cannot be processed, but Identity Manager should resubmit the event because it should be able to be processed soon. This status can be issued if your external application appears to be temporarily unavailable. However, this status should be used cautiously because a backlog results if Identity Manager continually retries one event.
IDMStatusError	The event cannot be processed and it should not be resubmitted. Identity Manager issues the error message and submits the next event. You should make a detailed error message so the problem can be corrected.
IDMStatusFatal	For some reason, the driver must be stopped. Identity Manager issues your message and stops the driver. This could be used if the external application appears to be permanently offline. The event remains in the queue and is resubmitted when the driver is restarted.

Writing Values

IDMSetCommand and/or a status subroutine must be called before specifying values with IDMWriteValues. IDMWriteValues (or its single-valued version IDMWriteValue) is used to set values to return to Identity Manager. It is passed a value name and an array of values. In the ADD_ASSOCIATION example above, IDMWriteValue is used to set the ASSOCIATION value. You can specify values for items listed in the table above.

Handling Query Events

For Query events, Identity Manager submits values that define the parameters of a search of the external application's identity management system. Queries are usually issued from the policies you have defined for your system. The table below specifies values that can be specified in queries. Not all values are relevant to your external application.

Table 5-21 *Query Values*

Value Name	Description
SCOPE	Specifies what identities will be searched. A base object is specified with the ASSOCIATION or DEST_DN values (see below). The value "entry" means that only the base object is searched. The value "subordinates" means that the immediate subordinates of the base object are searched. The value "subtree" (the default) indicates that the base object and all subordinates are searched. The last two values are only relevant in a hierarchical system.
ASSOCIATION	The base object for the search. If both ASSOCIATION and DEST_DN have values, ASSOCIATION is used. If neither is specified, the base object is the root of the identity management system.
DEST_DN	The base object for the search (see also ASSOCIATION above).
CLASS_NAME	The base class of the base object.
EVENT_ID	An identifier for the event.
SEARCH_CLASSES	A list of classes for which to search. Only identities of these classes are returned. If not specified, all identities in the scope matching SEARCH_ATTR_ values are returned (see below).
SEARCH_ATTRS	List of attribute names used in SEARCH_ATTR_ values (see below).
SEARCH_ATTR_*attr_name*	A value that the specified attribute must match. Replace *attr_name* with the desired attribute name. Only identities matching all SEARCH_ATTR_ filters are returned.
READ_ATTRS	List of attribute names whose values returned for each identiy match.
ALL_READ_ATTRS	The presence of this value indicates that all attribute values should be returned for matching identities.
NO_READ_ATTRS	The presence of this value indicates that no attribute values should be returned for matching identities.

Value Name	Description
READ_PARENT	The presence of this value indicates that the parent object of each matching identity should be returned. Only relevant in hierarchical systems.

Execute the query against the external application using application-provided tools. Then return each identity by setting an `INSTANCE` command, followed by relevant values from the list below.

Table 5-22 *Query Values*

Value Name	Description
CLASS_NAME	The class of the identity. Required.
SRC_DN	A distinguished name representing the logical location of the identity in the system (optional).
ASSOCIATION	The association of the identity, if available (optional).
PARENT	The association of the parent object of the identity (optional). Only relevant in hierarchical systems.
ATTR_*attr_name*	A list of values for the attribute specified by *attr_name*. Return attribute values specified by the READ_ATTRS value.

After returning all identities, call `IDMStatusSuccess` to indicate a successful query.

Subscriber Summary and Examples

Below is a more detailed summary of the actions to take for a non-Query event.

1. Gather information about the event using `IDMGetEventValues`. Return a warning or error if there is a problem.
2. Submit the event data to the external application using application-provided tools.
3. Set a command using `IDMSetCommand` and/or a status with the `IDMStatus` subroutines, based on the result of the event.
4. Set event values with `IDMWriteValues`.
5. If you have not already done so, set a status with an `IDMStatus` subroutine.

Below is an example of the `Add.vbs script`, which forms an association from an identity's CN and class name, and uses a hypothetical tool called `appadd`.

```
Sub ADD
  ClassName = IDMGetEventValue("CLASS_NAME")
  CN = IDMGetEventValue("CN")
  PhoneNumber = IDMGetEventValue("Telephone")
  If IsEmpty(ClassName) Or IsEmpty(CN) Then
    IDMStatusError "Add event: missing CLASS_NAME and/or CN"
  Else
    Command = "appadd -n """ & CN & """ -t """ & PhoneNumber & """"
    ExitCode = IDMExecute(Command)
    If ExitCode = 0 Then
      IDMSetCommand "ADD_ASSOCIATION"
      IDMWriteValue "ASSOCIATION", CN & ClassName
      IDMWriteValue "DEST_DN", IDMGetEventValue("SRC_DN")
      IDMStatusSuccess "Add event succeeded"
    Else
      IDMStatusError "Add event failed with error code " & ExitCode
    End If
  End If
End Sub
```

Handling a query is similar, except you return `INSTANCE` items rather than use other commands. Below is an example `Query.vbs` that searches an external application for a telephone number. It uses a hypothetical tool called `appsearch`.

```
Sub QUERY
  SearchName = IDMGetEventValue("SEARCH_ATTR_CN")
  If IsEmpty(SearchName) Then
    IDMStatusError "Query: no search value"
  Else
    Command = "appsearch -n " & SearchName
    Results = IDMExecuteIO(Command, Empty)
    If Results(0) = 0 Then
      Phone = Results(1)
      IDMSetCommand "INSTANCE"
      IDMWriteValue "CLASS_NAME", IDMGetEventValue("CLASS_NAME")
      IDMWriteValue "ASSOCIATION", IDMGetEventValue("ASSOCIATION")
      IDMWriteValue "ATTR_Telephone", Phone
      IDMStatusSuccess "Query succeeded"
    Else
      ' Return success with no results
      IDMStatusSuccess "Query succeeded (no matches)"
    End If
  End If
End Sub
```

Publisher Script Development

Events that occur on the external application are submitted to Identity Manager on the Publisher channel. The Scripting driver periodically polls the external application for events. How this poll detects events is implementation-specific and must be defined by you.

Polling for Application Events

The Driver calls `Poll.wsf` to detect application events. `Poll.wsf` should be implemented as follows:

1 Use application-provided tools to detect events in your application. (See Step 2.)

2 For each event:

  2a Call the `IDMPublishInit` function to set the appropriate command.

  2b Call `IDMPublishValues` to set event values.

  2c Call `IDMPublish` to submit the event to Identity Manager.

3 If there were events, call an `IDMStatus` function to report the status.

`IDMPublishInit` takes a command name as its single parameter. Below is a list of valid command names for `IDMPublishInit`.

Table 5-23 *Command Names*

Command Name	Description
ADD	Create an identity.
ADD_ASSOCIATION	Create an association for an identity.
DELETE	Remove an identity permanently.
MODIFY	Change an identity's attributes.
MODIFY_ASSOCIATION	Change an identity's association.
MODIFY_PASSWORD	Change an identity's password.
REMOVE_ASSOCIATION	Delete an identity's association.
RENAME	Change an identity's naming attribute.

Below is an example of a `Poll.wsf` that checks for a password change. It uses a hypothetical application tool called apppwd.

```
Results = IDMExecuteIO("apppwd --changes", Empty)
For I = 1 To UBound(Results)
  ' Entries are in the format "association=password"
  Association = Left(Results(I), InStr(Results(I), "=")-1)
  Password = Mid(Results(I), InStr(Results(I), "=")+1)
  IDMPublishInit "MODIFY_PASSWORD"
  IDMPublishValue "ASSOCIATION", Association
  IDMPublishValue "PASSWORD", Password

  IDMPublish
Next

IDMStatusSuccess "Poll succeeded"
```

Events submitted using `IDMPublish` are processed through your driver's Publisher channel policies. See the Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for more information.

Using the Heartbeat Script

Another script executed in the Publisher Channel is `Heartbeat.wsf.` This script is executed when the Publisher channel is idle for the interval specified in the driver parameters. (You can set the interval to 0 so no heartbeat is issued.) You can use the heartbeat to check the availability of the external system or do “idle state” tasks. The IDMHeartbeatSuccess, IDMHeartbeatWarning, and IDMHeartbeatError subroutines can be used to indicate the result of the heartbeat. Below is an example based on a hypothetical tool called apphealth.

```
ExitCode = IDMExecute("apphealth")
If ExitCode = 0 Then
  IDMHeartbeatSuccess "Heartbeat succeeded"
Else
  IDMHeartbeatError "Heartbeat failed with error code " & ExitCode
End If
```

The response to the heartbeat is implementation-dependent, and can be defined in policies or in the script itself. You could send a message to auditing using NetIQ Audit. You could store a value in a file, and have Subscriber scripts read the file and call IDMStatusRetry if they find that value in the file.

Other Scripting Topics

- “Driver Parameters” on page 99
- “Querying the Identity Vault” on page 100
- “Tracing and Debugging” on page 101

Driver Parameters

A driver has values known as driver parameters. The driver parameters are divided into driver settings applicable to the whole driver, and Subscriber and Publisher Settings for their respective channels. The IDMGetDriverParam, IDMGetSubscriberParam, and IDMGetPublisherParam functions can be used to retrieve these values. The table below shows parameters in the default Scripting driver. Other parameters can be added to the driver’s XML Configuration file (see “Managing Identity Manager Drivers” in the *NetIQ Identity Manager Administration Guide*).

Table 5-24 *Driver Parameters*

Parameter Name	Driver/Channel	Description	Values
INSTALL_PATH	Driver	The installation path of the driver	string value
auto-loopback-detection	Driver	Whether to enable automatic loopback detection	true/false
script-command	Driver	The command to use to execute scripts	string value
script-trace-file	Driver	The file, relative to the driver installation path, to which to write trace messages	string value

Parameter Name	Driver/Channel	Description	Values
subscriber-script	Subscriber	The script file for Subscriber events, relative to the driver installation path	string value
polling-script	Publisher	The script file that runs when the Publisher polls for application events	string value
heartbeat-script	Publisher	The script file that runs when the Publisher checks application status	string value
pub-polling-interval	Publisher	The interval in seconds between Publisher polls for application events	number
pub-heartbeat-interval	Publisher	The amount of idle time in seconds before a heartbeat event is issued	number

In the following example, a script retrieves the Publisher polling interval.

```
PollingInterval = IDMGetPublisherParam("pub-polling-interval")
```

Querying the Identity Vault

Scripts might need to retrieve information from the Identity Vault. On the Subscriber channel only, they can do this by issuing a query.

1. Initialize the query with IDMQueryInit.
2. Set query search parameters using functions listed below.

Function	Description
IDMQuerySetAssociation(Association)	Sets the association of the object to query.
IDMQuerySetSearchRoot(SearchRoot)	Sets the DN (in slash format) of the object to query.
IDMQueryAddSearchAttr(Name, Value)	Specifies a search condition of the form Name=Value. The query will only return an instance if the named attribute has a value matching Value.
IDMQueryAddReadAttr(Name)	Specifies an attribute whose values will be returned with the instance.
IDMQuerySetReadParent(ReadParent)	Specifies whether the association and DN of the parent of the queried object should be returned (False by default).

3. Execute the query with IDMQuery.
4. Read the result (instance) using functions listed below.

Currently query support is limited. It will only return one instance based on the specified association or DN. (If both association and DN are specified, association is used.) The functions below allow you to retrieve information from the instance.

Function	Description
IDMGetQueryInstanceAssociation	Returns the association of the instance.
IDMGetQueryInstanceDN	Returns the DN of the instance (in slash format).
IDMGetQueryInstanceClass	Returns the class name of the instance.
IDMGetQueryInstanceParentAssociation	Returns the association of the parent of the instance (if requested).
IDMGetQueryInstanceParentDN	Returns the DN of the parent of the instance (if requested).
IDMGetQueryInstanceAttrNames	Returns an array containing the names of the attributes retrieved for the instance. Returns Empty if no attributes were retrieved.
IDMGetQueryInstanceAttrCount	Returns the number of attributes retrieved for the instance.
IDMGetQueryInstanceAttrValues(AttrName)	Returns an array of values for the attribute AttrName. Returns Empty if no values are available.
IDMGetQueryInstanceAttrValue(AttrName)	Returns a string value for the attribute AttrName. If multiple values are available, the first one is returned. Returns Empty if no values are available.

The following is an example of a query that retrieves an object's address and postal code.

```
IDMQueryInit
IDMQuerySetAssociation IDMGetEventValue("ASSOCIATION")
IDMQueryAddReadAttr "SA"          ' Street Address
IDMQueryAddReadAttr "Postal Code"

If IDMQuery Then
  Address = IDMQueryGetInstanceAttrValue("SA")
  ZIPCode = IDMQueryGetInstanceAttrValue("Postal Code")
  ' ... etc. ...
End If
```

Tracing and Debugging

The IDMTrace function allows you to write a message to the Script Trace File specified in the Driver Parameters. Tracing is useful for script debugging and auditing.

```
IDMTrace "Trace message"
```

When developing scripts, you might need to do some debugging to track down problems. The following list indicates some facilities for debugging.

- The Driver traces activity to its Trace file (`logs\trace.log` by default). The trace level setting in `conf\wsdrv.conf` controls how much debugging is written to the log.

Trace Level	Description
0	No debugging.
1-3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous level plus Remote Loader, driver, driver shim, and Driver connection messages.
5-7	Previous level plus Change Log and loopback messages. Higher trace levels provide more detail.
8	Previous level plus Driver status log, Driver parameters, Driver command line, Driver security, Driver Web server, Driver schema, Driver encryption, Driver SOAP API, and Driver include/exclude file messages.
9	Previous level plus low-level networking and operating system messages.
10	Previous level plus maximum low-level program details.

  The trace level is set using the -trace option in `wsdrv.conf,` for example. -trace 9.

  You can view the trace file through a Web browser:

  1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
  2. Authenticate by using any username and the password that you specified as the Remote Loader password.
  3. Click **Trace**.

- The IDMTrace function described above writes output to the trace file specified in the Driver Parameters (`logs\script-trace.log` by default).
- The eDirectory tool DSTrace can be used to monitor Identity Manager activity. Set the tracing level for the Driver in Identity Console. DSTrace shows the XML documents being submitted to the Driver for events, and how Policies are evaluated. It also shows the status and message for each event.
- The Status Log is written to `logs\dirxml.log.` It shows a summary of the events that have been recorded on the Subscriber and Publisher channels.

  You can view the Status Log through a Web browser:

  1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
  2. Authenticate by using any username and the password that you specified as the Remote Loader password.
  3. Click **Status**.

Although it is best to run the driver as a service in production environments, you can run `wsdriver.exe` as a standard program. When you do so, a console window displays trace messages (see above) for the driver. Also, text written to standard output from scripts (such as using `WScript.Echo` in VBScript) is displayed in this window.

VBScript programs can be debugged using programs such as Microsoft Visual Studio and Microsoft Script Debugger. Change the Script Command driver parameter to use the //x option: cscript //nologo //x. When the driver shim executes a script, you are prompted to debug the script execution.

Deployment

The Scripting driver is installed by using a setup program. See "Installing the Windows Scripting Driver" on page 20 for information on installing the default driver.

Deploying a Custom Driver

To deploy your custom driver, the end user should first run the Windows Scripting driver installation program provided by the installation media (see "Installing the Windows Scripting Driver" on page 20 for more information.) This program installs core files needed by the driver. Then, your custom driver files can be deployed in any convenient way, whether through an installation program or even simply an archive file. The table below shows the directory structure below the installation directory and what files are installed.

Table 5-25 *Directory Structure and Files*

Directory	Description	Required Files

`bin\`	Location of executable programs	`EventReader.exe`
		`idmevent.exe`
		`wsdriver.exe`

`changelog\`	Used for Publisher event processing	None
`conf\`	Location of the driver shim configuration file	`wsdrv.conf` (customized)
`keys\`	Location of security key files	None
`logs\`	Location of log files	None
`loopback\`	Used for automatic loopback detection	None
`rules\`	Location of Driver configuration file	`Scripting.xml` (customized)
`schema\`	Location of schema files	`schema.def` (customized)
`scripts\`	Location of script files	Those required by your Driver (customized)

If you are using an installation program, you can obtain the driver's installation path from the following registry value:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Novell\Windows Script Driver\Path
```

Or for the x86 driver running an x64 version of Windows, use the path from the following registry value:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Novell
\Windows Script Driver\Path
```

The formats of `wsdrv.conf` and `schema.def` can be viewed in "The Driver Shim Configuration File" on page 32 and "The Connected System Schema File" on page 38.

If SSL encryption is desired for communication between the driver shim and Identity Manager engine, a certificate must be retrieved from the Identity Vault. Run the following command and follow the prompts to retrieve the certificate:

```
wsdriver.exe -s
```

The certificate will be stored in the `keys\` directory. You must have LDAP with SSL available for the Metadirectory. When making an installation program for deployment, you might want to include this command as part of the installation:

To ensure that only authorized systems access the Metadirectory, a Driver Object password and Remote Loader password are used. Run the following command and enter the passwords at the prompts:

```
wsdriver.exe -sp
```

This action also can be incorporated into an installation program.

You should run a Scripting driver as a service. To the install the service, run the following command or include it as part of an installation program:

```
wsdriver.exe -installService
```

The service, which can then be run from the Services applet, can be removed as follows:

```
wsdriver.exe -removeService
```

You should distribute the XML configuration file that contains parameters and policies your driver needs. The user can then select it when installing your driver.

Windows PowerShell Developer Guide

Windows PowerShell is a new command-line shell and scripting environment intended for system administrators. Part of Microsoft .NET technology, PowerShell allows you to manage and configure any aspect of Windows.

The Scripting driver provides a complete Windows PowerShell API for interacting with identity management systems whose tools (including APIs) are available on Windows. The Identity Vault and Identity Manager can run on any supported operating system. Identity Manager can communicate with any supported system on which the driver is installed via an encrypted network connection.

Before beginning script development, review the preceding topics in this section for information on defining what data is synchronized between identity management systems.

With additional development work, the driver can also be adapted to support any scripting language that supports command line operation.

Developing a custom driver with PowerShell is discussed in this section. Topics include

- "Application Tools Evaluation" on page 105
- "Policy and Script Development" on page 106
- "Deployment" on page 118

Application Tools Evaluation

To change the data in your external application, you need to know how to use the application's tools or API (Application Programming Interface). These tools must provide automated operation and not require user input.

Application Command Line Tools

Applications often include *command-line* tools. These tools are manually executed from the Windows command prompt, and they can be executed from scripts. For example, suppose an application provides a tool to add identities with a program called `appadd.exe`.

```
appadd -n "Bob Smith" -t "818-555-2100"
```

This command adds an identity named "Bob Smith" with the specified phone number. The strings following the program name are called *parameters* or *arguments*. Because PowerShell is a command shell, commands can be executed by the script code:

```
   $name = "Bob Smith
   $phone = "818-555-2100"
   appadd -n $name -t $phone
   $exitcode = $LASTEXITCODE
```

You can use the pipeline to send data to a program's standard input and to receive output from the program's standard output and standard error.

```
  # Create an array with two lines
  $inlines = "USERNAME=Bob Smith", "TELEPHONE=818-555-2100"
  $inlines | appadd | Set-Variable outlines
```

PowerShell will output the contents of the inlines array to appadd's standard input, one item per line. The (standard) output will be stored in outlines, as a single string variable for one line or an array for more than one line. (You don't use the $ character with the Set-Variable cmdlet.)

Thus, with command-line tools, you can construct the command line's parameters using the values passed to the script, then execute the program.

Application Objects

Another way to modify application data is through Windows COM (Common Object Model) or .NET objects. Consult your application's documentation to see whether it exposes any COM or .NET objects. These objects can be loaded directly in PowerShell:

```
$appobject = New-Object -comobject MyApplication.MyObject
$appobject.AddIdentity("Bob Smith", "818-555-2100")
```

Note that there are no guarantees regarding types and availability of tools. You must determine if sufficient tools are provided by the application. If they are not, you can contact the application's developers and request them.

Make a list of the tools that can be used for each event type. The application might provide multiple tools or a single program that can be used for any event type.

Application Event Monitoring

You also need to determine what tools are available for monitoring event changes in the application. The Scripting driver works on a polling system. It periodically calls a polling script to determine what has changed in the external application. Here are some ideas for monitoring changes:

- The first time the polling script is run, a list of identities and relevant attributes is read from the application using an application-provided tool. This list is stored as a file. On subsequent polls, a new list is generated and compared to the old list. Any differences are submitted as events to the driver.
- The application provides a tool that allows you to request all identities that have changed after a certain point in time. The polling script requests events that have occurred since the previous poll.
- The application allows a script to be run when an event occurs. You write a script that stores the event data into a file. When the (Scripting driver) polling script runs, it consumes this file and submits the data as an event to the driver.

Monitoring the application's changes might be the most difficult aspect of developing your driver. You must study your application's tools to determine the best way to achieve synchronization.

Policy and Script Development

At this point you should have a list of what data will be synchronized, how events will be handled, and what application tools are available. It is time to develop the heart of your driver in policies and scripts.

Many types of tasks can be handled in driver policies. You can import the driver configuration provided with the Scripting driver, and then edit policies in NetIQ Identity Console. You can also edit policies and simulate their operation in NetIQ Designer. The extensive functionality of policies is outside the scope of this document, so you should refer to your Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for help.

Tasks that don't interact with the external application might be more suited to policies. On the other hand, if you are more familiar with your scripting language than policies, you can develop your driver more quickly by using scripts.

Event Data Format

Event data is submitted to the scripts in *name/value pair* format. This format consists of lines containing a name, an equal sign (=) and a value. Therefore, each line is a name/value pair. Each name/value pair is unique, but there can be multiple name/value pairs with identical names but different values.

```
ASSOCIATION=BobUser
ADD_TELEPHONE=818-555-2100
ADD_TELEPHONE=818-555-9842
```

You typically don't need to worry about the format. The script library provides functions for retrieving event data.

Subscriber Script Development

After all policy processing is complete, Identity Manager submits the event in XML format to the driver shim. The driver shim submits the event data to the scripts.

In the default Scripting driver, the script `Subscriber.ps1` in the `scripts` folder is called. This script does some preliminary processing, and then calls a routine from an included script. The included scripts correspond to the Subscriber event types: `Add.ps1`, `Modify.ps1`, `ModifyPassword.ps1`, `Delete.ps1`, `Rename.ps1`, `Move.ps1`, and `Query.ps1`.

For each event type, you should retrieve the information you need from the event data, submit changes to the external application using application-provided tools and return a status (such as success or failure) to Identity Manager.

Event data is retrieved primarily using the idm_geteventvalues function. This function returns an array of values corresponding to the name specified as the function's parameter. (idm_geteventvalue is available for single-valued items.) The following table shows many item names.

Table 5-26 *Item Names*

Value	Description
`COMMAND`	The command for the event, usually indicating the event type.
`ASSOCIATION`	The identifier that distinguishes an identity on both identity management systems.
`CLASS_NAME`	An identity's class, such as User or Group.
`SRC_DN`	An identity's distinguished name (DN) in the namespace of the source (sender), in slash format.
`EVENT_ID`	An identifier for the event, for internal use.
`SRC_ENTRY_ID`	An identifier for the identity that generated the event, in the namespace of the source (sender).
`DEST_DN`	An identity's distinguished name (DN) in the namespace of the destination (receiver), in slash format.

Value	Description
DEST_ENTRY_ID	An identifier for an entry in the namespace of the destination (receiver).
ADD_*attr_name*	A value to be added to an identity, for attribute *attr_name*.
REMOVE_*attr_name*	A value to be removed from an identity, for attribute *attr_name*.
ADD_REF_*attr_name*	A value to be added to attribute *attr_name*, where the value is an association referring to another identity.
REMOVE_REF_*attr_name*	A value to be removed from attribute *attr_name*, where the value is an association referring to another identity.
OLD_PASSWORD	The previous password for an identity that has changed its password. Used in Modify Password events.
PASSWORD	The new password for an identity. Used in Add and Modify Password events.
OLD_SRC_DN	The distinguished name of an identity before a Move or Rename event.
REMOVE_OLD_NAME	Specifies whether an old relative distinguished name should be deleted or retained. Used in Rename events.
STATUS_LEVEL	The status of an event: success, warning, retry, error or fatal.
STATUS_MESSAGE	A message to report with a status.
STATUS_TYPE	A type of status, such as heartbeat.

Handling Associations

The association value indicates which identity has been changed. If the identity has no association, an association must be generated for it using an implementation-specific rule that you have adopted. When Identity Manager processes an event for an identity with no association, it executes the driver's Matching policy. This policy attempts to match the event's identity to an identity on the external application's system. Doing this usually involves executing a query. The default Matching policy included with the Scripting driver queries for matching Users and Groups based on the CN attribute. If the event's identity matches an identity on the external application, both identities must be assigned the new association. Assigning this association can be included as part of the query-handling script. (Handling queries is discussed in "Handling Query Events" on page 52.) If no identity matches, an Add event is issued, and the new association can be assigned as part of the Add event-handling script:

```
idm_setcommand "ADD_ASSOCIATION"
idm_writevalue "ASSOCIATION" $my_association
           (etc.)
```

The example above also illustrates the idm_setcommand function. This function sets a command value that indicates what action Identity Manager should perform. The ADD_ASSOCIATION command assigns an association to an identity, as discussed above. The Subscriber can also issue MODIFY_ASSOCIATION and REMOVE_ASSOCIATION commands.

Returning an Event Status

On the Subscriber channel, you often do not need Identity Manager to perform an action, but simply need to report a status. The IDMStatus subroutines noted below can be used to indicate a status to Identity Manager. They take a message to be logged as the parameter.

Table 5-27 *Subroutines*

Subroutine	Identity Manager Action
idm_statussuccess	Identity Manager marks the event as a success and submits the next event in the event queue, if any. You should issue this status even if your script does nothing.
idm_statuswarning	The event can be processed, but it might require attention. Identity Manager issues your warning message in its log, and then submits the next event.
idm_statusretry	The event cannot be processed, but Identity Manager should resubmit the event because it should be able to be processed soon. This status can be issued if your external application appears to be temporarily unavailable. However, this status should be used cautiously because a backlog results if Identity Manager continually retries one event.
idm_statuserror	The event cannot be processed and it should not be resubmitted. Identity Manager issues the error message and submits the next event. You should make a detailed error message so the problem can be corrected.
idm_statusfatal	For some reason, the driver must be stopped. Identity Manager issues your message and stops the driver. This could be used if the external application appears to be permanently offline. The event remains in the queue and is resubmitted when the driver is restarted.

Writing Values

idm_setcommand and the status functions must be issued before specifying values with idm_writevalues. idm_writevalues (or its single-valued version idm_writevalue) is used to set values to return to Identity Manager. It is passed a value name and an array of values. In the `ADD_ASSOCIATION` example above, idm_writevalue is used to set the `ASSOCIATION` value. You can specify values for items listed in the table above.

Handling Query Events

For Query events, Identity Manager submits values that define the parameters of a search of the external application's identity management system. Queries are usually issued from the policies you have defined for your system. The table below specifies values that can be specified in queries. Not all values are relevant to your external application.

Table 5-28 Query Values

Value Name	Description
SCOPE	Specifies what identities will be searched. A base object is specified with the ASSOCIATION or DEST_DN values (see below). The value “entry” means that only the base object is searched. The value “subordinates” means that the immediate subordinates of the base object are searched. The value “subtree” (the default) indicates that the base object and all subordinates are searched. The last two values are only relevant in a hierarchical system.
ASSOCIATION	The base object for the search. If both ASSOCIATION and DEST_DN have values, ASSOCIATION is used. If neither is specified, the base object is the root of the identity management system.
DEST_DN	The base object for the search (see also ASSOCIATION above).
CLASS_NAME	The class of the base object.
EVENT_ID	An identifier for the event.
SEARCH_CLASSES	A list of classes for which to search. Only identities of these classes are returned. If not specified, all identities in the scope matching SEARCH_ATTR_ values are returned (see below).
SEARCH_ATTRS	A list of attribute names used in SEARCH_ATTR_ values (see below).
SEARCH_ATTR_*attr_name*	A value that the specified attribute must match. Replace *attr_name* with the desired attribute name. Only identities matching all SEARCH_ATTR_ filters are returned.
READ_ATTRS	A list of attribute names whose values are returned for each matching identity.
ALL_READ_ATTRS	The presence of this value indicates that all attribute values should be returned for matching identities.
NO_READ_ATTRS	The presence of this value indicates that no attribute values should be returned for matching identities, only associations.
READ_PARENT	The presence of this value indicates that the parent object of each matching identity should be returned. Only relevant in hierarchical systems.

Execute the query against the external application using application-provided tools. Then return each identity by setting an INSTANCE command, followed by relevant values from the list below.

Table 5-29 Query Values

Value Name	Description
CLASS_NAME	The class of the identity. Required.
SRC_DN	A distinguished name representing the logical location of the identity in the system (optional).
ASSOCIATION	The association of the identity, if available (optional).
PARENT	The association of the parent object of the identity (optional). Only relevant in hierarchical systems.
ATTR_*attr_name*	A list of values for the attribute specified by *attr_name*. Return attribute values specified by the READ_ATTRS value.

After returning all identities, call idm_statussuccess to indicate a successful query.

Subscriber Summary and Examples

Below is a more detailed summary of the actions to take for a non-Query event.

1. Gather information about the event using idm_geteventvalues. Return a warning or error if there is a problem.
2. Submit the event data to the external application using application-provided tools.
3. Set a command using idm_setcommand and/or a status with the idm_status subroutines, based on the result of the event.
4. Set event values with idm_writevalues.
5. If you have not already done so, set a status with an idm_status subroutine.

Below is an example of the Add.ps1 script, which forms an association from an identity's CN and class name, and uses a hypothetical tool called appadd.

```
function idm_add
{
  $classname = idm_geteventvalue "CLASS_NAME"
  $cn = idm_geteventvalue "CN"
  $phone = idm_geteventvalue "Telephone"
  if ($classname -eq "" -or $cn -eq "") {
    idm_statuserror "Add event: missing CLASS_NAME and/or CN"
  }
  else {
    appadd -n "$cn" -t "$phone"
    if ($LASTEXITCODE -eq 0) {
      idm_setcommand "ADD_ASSOCIATION"
      idm_writevalue "ASSOCIATION" "$cn$classname"
      idm_writevalue "DEST_DN" (idm_geteventvalue "SRC_DN")
      idm_statussuccess "Add event succeeded"
    }
    else {
      idm_statuserror "Add event failed with error code $LASTEXITCODE"
    }
  }
}
```

Handling a query is similar, except you return INSTANCE items rather than use other commands. Below is an example `Query.ps1` that searches an external application for a telephone number. It uses a hypothetical tool called appsearch.

```
function idm_query
{
  $searchname = idm_geteventvalue "SEARCH_ATTR_CN"
  if ($searchname -eq "") {
    idm_statuserror "Query: no search value"
  }
  else {
    # the telephone number is output from the application
    appsearch -n "$searchname" | Set-Variable phone
    if ($phone -ne $null) {
      idm_setcommand "INSTANCE"
      idm_writevalue "CLASS_NAME" (idm_geteventvalue "CLASS_NAME")
      idm_writevalue "ASSOCIATION" (idm_geteventvalue "ASSOCIATION")
      idm_writevalue "ATTR_Telephone" $phone
      idm_statussuccess "Query succeeded"
    }
    else {
      # Return success with no results
      idm_statussuccess "Query succeeded (no matches)"
    }
  }
}
```

Publisher Script Development

Events that occur on the external application are submitted to Identity Manager on the Publisher channel. The Scripting driver periodically polls the external application for events. How this poll detects events is implementation-specific and must be defined by the user.

Polling for Application Events

The driver calls `Poll.ps1` to detect application events. `Poll.ps1` should be implemented as follows:

1. Use application-provided tools to detect events in your application.
2. For each event:
   - **2a** Call the idm_publishinit function to set the appropriate command.
   - **2b** Call idm_publishvalues to set event values.
   - **2c** Call idm_publish to submit the event to Identity Manager.
3. If there were events, call an idm_status function to report the status.

idm_publishinit takes a command name as its single parameter. Below is a list of valid command names for idm_publishinit.

Table 5-30 *Command Names*

Command Name	Description
ADD	Create an identity.
ADD_ASSOCIATION	Create an association for an identity.
DELETE	Remove an identity permanently.
MODIFY	Change an identity’s attributes.
MODIFY_ASSOCIATION	Change an identity’s association.
MODIFY_PASSWORD	Change an identity’s password.
REMOVE_ASSOCIATION	Delete an identity’s association.
RENAME	Change an identity’s naming attribute.

Below is an example of a `Poll.ps1` that checks for a password change. It uses a hypothetical application tool called apppwd.

```
apppwd --changes | Set-Variable results
foreach ($result in $results) {
  # Entries are in the format "association=password"
  $tokens = $result.split("=")
  $association = $tokens[0]
  $password = $tokens[1]
  idm_publishinit "MODIFY_PASSWORD"
  idm_publishvalue "ASSOCIATION" $association
  idm_publishvalue "PASSWORD" $password

  idm_publish
next

idm_statussuccess "Poll succeeded"
```

Events submitted using idm_publish are processed through your driver's Publisher channel policies. See the Identity Manager policy guides on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/) for more information.

Using the Heartbeat Script

Another script executed in the Publisher Channel is `Heartbeat.ps1.` This script is executed when the Publisher channel is idle for the interval specified in the driver parameters. (You can also set the interval to 0 so no heartbeat is issued.) You can use the heartbeat to check the availability of the external system or do "idle state" tasks. The idm_heartbeatsuccess, idm_heartbeatwarning, and idm_heartbeaterror subroutines can be used to indicate the result of the heartbeat. Below is an example based on a hypothetical tool called apphealth.

```
apphealth
if ($LASTEXITCODE -eq 0) {
  idm_heartbeatsuccess "Heartbeat succeeded"
}
else {
  idm_heartbeaterror "Heartbeat failed with error code $LASTEXITCODE"
}
```

The response to the heartbeat is implementation-dependent, and can be defined in policies or in the script itself. You could send a message to auditing using NetIQ Audit. You could store a value in a file and have Subscriber scripts read the file and call idm_statusretry if they find that value in the file.

Other Scripting Topics

- "Driver Parameters" on page 114
- "Querying the Identity Vault" on page 115
- "Tracing and Debugging" on page 117

Driver Parameters

A driver has values known as driver parameters. The driver parameters are divided into driver settings applicable to the whole driver, and Subscriber and Publisher settings for their respective channels. The idm_getdriverparam, idm_getsubscriberparam, and idm_getpublisherparam functions can be used to retrieve these values. The table below shows parameters in the default Scripting driver. Other parameters can be added to the driver's XML Configuration file (see "Managing Identity Manager Drivers" in the *NetIQ Identity Manager Administration Guide*).

Table 5-31 *Driver Parameters*

Parameter Name	**Driver/Channel**	**Description**	**Values**
INSTALL_PATH	Driver	The installation path of the driver	string value
auto-loopback-detection	Driver	Whether to enable automatic loopback detection	true/false

Parameter Name	Driver/Channel	Description	Values
script-command	Driver	The command to use to execute scripts	string value
script-trace-file	Driver	The file, relative to the driver installation path, to which to write trace messages	string value
subscriber-script	Subscriber	The script file for Subscriber events, relative to the driver installation path	string value
polling-script	Publisher	The script file that runs when the Publisher polls for application events	string value
heartbeat-script	Publisher	The script file that runs when the Publisher checks application status	string value
pub-polling-interval	Publisher	The interval in seconds between Publisher polls for application events	number
pub-heartbeat-interval	Publisher	The amount of idle time in seconds before a heartbeat event is issued	number

In the following example, a script retrieves the Publisher polling interval.

```
$pollinginterval = idm_getpublisherparam("pub-polling-interval")
```

Querying the Identity Vault

Scripts might need to retrieve information from the Identity Vault. They can do this by issuing a query.

1. Initialize the query with idm_queryinit.
2. Set query search parameters using functions listed below.
3. Execute the query with idm_doquery.
4. Read the result (instance) using functions listed below.

This table lists functions for setting query search parameters.

Function	Description
idm_querysetassociation($association)	Sets the association of the object to query.
idm_querysetsearchroot($searchroot)	Sets the DN (in slash format) of the object to query.
idm_queryaddsearchattr($name, $value)	Specifies a search condition of the form $name=$value. The query will only return an instance if the named attribute has a value matching $value.
idm_queryaddreadattr($name)	Specifies an attribute whose values will be returned with the instance.
idm_querysetreadparent($readparent)	Specifies whether the association and DN of the parent of the queried object should be returned ($False by default).

Currently query support is limited. It will only return one instance based on the specified association or DN. (If both association and DN are specified, association is used.) The functions below allow you to retrieve information from the instance.

Function	Description
idm_getqueryinstanceassociation	Returns the association of the instance.
idm_getqueryinstancedn	Returns the DN of the instance (in slash format).
idm_getqueryinstanceclass	Returns the class name of the instance.
idm_getqueryinstanceparentassociation	Returns the association of the parent of the instance (if requested).
idm_getqueryinstanceparentdn	Returns the DN of the parent of the instance (if requested).
idm_getqueryinstanceattrnames	Returns an array containing the names of the attributes retrieved for the instance. Returns Empty if no attributes were retrieved.
idm_getqueryinstanceattrcount	Returns the number of attributes retrieved for the instance.
idm_getqueryinstanceattrvalues($attrname)	Returns an array of values for the attribute $attrname. Returns Empty if no values are available.
idm_getqueryinstanceattrvalue($attrname)	Returns a string value for the attribute $attrname. If multiple values are available, the first one is returned. Returns Empty if no values are available.

Following is an example of a query of the Identity Vault that retrieves an object’s address and postal code.

```
idm_queryinit
idm_querysetassociation (idm_geteventvalue "ASSOCIATION")
idm_queryaddreadattr "SA"          # Street Address
idm_queryaddreadattr "Postal Code"

$result = idm_doquery
if ($result) {
  $address = idm_querygetinstanceattrvalue "SA"
  $zipcode = idm_querygetinstanceattrvalue "Postal Code"
  # ... etc. ...
}
```

Tracing and Debugging

The function idm_trace allows you to write a message to the Script Trace File specified in the Driver Parameters. Tracing is useful for script debugging and auditing.

```
idm_trace "Trace message"
```

If the driver shim is being run as a Windows Service, all output from scripts will also be recorded in the Script Trace File.

When developing scripts, you might need to do some debugging to track down problems. The following list indicates some facilities for debugging.

- The driver traces activity to its Trace file (`logs\trace.log` by default). The trace level setting in `conf\wsdrv.conf` controls how much debugging is written to the log.

Trace Level	Description
0	No debugging.
1-3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous level plus Remote Loader, driver, driver shim, and driver connection messages.
5-7	Previous level plus Change Log and loopback messages. Higher trace levels provide more detail.
8	Previous level plus driver status log, driver parameters, driver command line, driver security, driver Web server, driver schema, driver encryption, driver SOAP API, and driver include/exclude file messages.
9	Previous level plus low-level networking and operating system messages.
10	Previous level plus maximum low-level program details.

The trace level is set using the `-trace` option in `wsdrv.conf`, for example, `-trace 9`.

You can view the trace file through a Web browser:

1. Use a Web browser to access the driver shim at `https://`*`driver-address`*`:8091`. Substitute the DNS name or IP address of your driver for *driver-address*.

2. Authenticate by using any user name and the password that you specified as the Remote Loader password.
3. Click **Trace**.

- The idm_trace function described above writes output to the trace file specified in the driver parameters (`logs\script-trace.log` by default).
- The eDirectory tool DSTrace can be used to monitor Identity Manager activity. Set the tracing level for the driver in Identity Console. DSTrace shows the XML documents being submitted to the driver for events, and how Policies are evaluated. It also shows the status and message for each event.
- The Status Log is written to `logs\dirxml.log.` It shows a summary of the events that have been recorded on the Subscriber and Publisher channels.

  You can view the Status Log through a Web browser:

  1. Use a Web browser to access the driver shim at `https://`*`driver-address`*`:8091`. Substitute the DNS name or IP address of your driver for *driver-address*.
  2. Authenticate by using any user name and the password that you specified as the Remote Loader password.
  3. Click **Status**.

Although it is best to run the driver as a service in production environments, you can run `wsdriver.exe` as a standard program. When you do so, a console window displays trace messages (see above) for the driver. Also, text written to standard output from scripts is displayed in this window.

Deployment

The Scripting driver is installed by using a setup program. See "Installing the Windows Scripting Driver" on page 20 for information on installing the default driver.

Deploying a Custom Driver

To deploy your custom driver, the end user should first run the Windows Scripting driver installation program provided by the installation media (see "Installing the Windows Scripting Driver" on page 20). This program installs core files needed by the driver. Then your custom driver files can be deployed easily, whether through an installation program or an archive file. The table below shows the directory structure below the installation directory and what files are installed.

Table 5-32 *Directory Structure and Files*

Directory	Description	Required Files

`bin\`	Location of executable programs	`EventReader.exe`
		`idmevent.exe`
		`wsdriver.exe`

`changelog\`	Used for Publisher event processing	None

Directory	Description	Required Files
`conf\`	Location of the driver shim configuration file	`wsdrv.conf` (customized)
`keys\`	Location of security key files	None
`logs\`	Location of log files	None
`loopback\`	Used for automatic loopback detection	None
`rules\`	Location of driver configuration file	`Scripting.xml` (customized)
`schema\`	Location of schema files	`schema.def` (customized)
`scripts\powershell\`	Location of script files	Those required by your Driver (customized)

If you are using an installation program, you can obtain the driver's installation path from the following registry value:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Novell\Windows Script Driver\Path
```

Or for the x86 driver running an x64 version of Windows, use the path from the following registry value:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Novell
\Windows Script Driver\Path
```

The formats of `wsdrv.conf` and `schema.def` can be viewed in "The Driver Shim Configuration File" on page 32 and "The Connected System Schema File" on page 38.

If SSL encryption is desired for communication between the driver shim and Identity Manager engine, a certificate must be retrieved from the Identity Vault. Run the following command and follow the prompts to retrieve the certificate:

```
wsdriver.exe -s
```

The certificate will be stored in the `keys\` directory. You must have LDAP with SSL available for the Metadirectory. When making an installation program for deployment, you might want to include this command as part of the installation:

To ensure that only authorized systems access the Metadirectory, a Driver Object password and Remote Loader password are used. Run the following command and enter the passwords at the prompts:

```
wsdriver.exe -sp
```

This action also can be incorporated into an installation program.

You should run a Scripting driver as a service. To the install the service, run the following command or include it as part of an installation program:

```
wsdriver.exe -installService
```

The service, which can then be run from the Services applet, can be removed as follows:

```
wsdriver.exe -removeService
```

You should distribute the XML configuration file that contains parameters and policies your driver needs. The user can then select it when installing your driver.

Using an Alternate Scripting Language

An alternative scripting language can be used by porting the globals, IDMLib, and other scripts to the alternative language. Porting requires a solid understanding of one of the provided languages (Bourne Shell, Perl and Microsoft VBScript and PowerShell) and the target language. The language should have facilities for executing programs and reading/writing to their standard input and output. The script language program must be able to be run from the command prompt.

To change the program the driver executes when running a script, modify the Script Command driver parameter. After the name of the program, you can include any needed command line parameters. To this string, the driver appends a space and the name of the Subscriber, Polling or Heartbeat script. For the Subscriber script, the full path to the event file and the full path to the Driver Parameter file are the next two parameters. For the Polling and Heartbeat scripts, the Driver Parameter file is the next parameter.

The event file contains the event document in XML format. On Windows, you must use the `EventReader.exe` program included with the Scripting driver to process this file. Running EventReader with the nvpairs option retrieves the event document as name/value pairs (the default). Using the xds option retrieves the event in its original XDS/XML format. On Linux and UNIX, events are submitted to scripts using shared memory. You must use the Shared Memory Helper, ussmh, to process these events. Running ussmh with the -xml option retrieves the XML document in its original XDS/XML format.

The Driver Parameter file contains name/value pairs for the driver parameters. See "UNIX Shell Developer Guide" on page 46, "Perl Developer Guide" on page 61, and "Microsoft VBScript Developer Guide" on page 90 for more information.

On the Publisher channel, you must use the Change Log tool (`idmevent.exe` on Windows and `usclh` on Linux/UNIX), to submit events to the driver.

The names of the Subscriber, Polling and Heartbeat scripts can be altered in the driver parameters.

6 Using the Scripting Driver

This section provides information about operational tasks commonly used with the Identity Manager driver for Scripting.

Topics include

Starting and Stopping the Driver

To start the driver:

1. Click the **IDM Administration** tab from the Identity Console home screen.
2. Click the specific driver set object on the right hand side of your computer screen to display all the drivers associated to it.
3. Click the **Actions** icon on the specific driver and select **Start Driver**.

To stop the driver:

1. Click the **IDM Administration** tab from the Identity Console home screen.
2. Click the specific driver set object on the right hand side of your computer screen to display all the drivers associated to it.
3. Click the **Actions** icon on the specific driver and select **Stop Driver**.

Starting and Stopping the Driver Shim

To start the driver shim, perform the task appropriate for your operating system as shown in the following table:

Table 6-1 *Starting the Driver Shim*

Operating System	Command/Task
AIX	`/etc/rc.d/init.d/usdrvd start`
FreeBSD	`/usr/local/etc/rc.d/usdrvd start`
HP-UX	`/sbin/init.d/usdrvd start`
Linux	`/etc/init.d/usdrvd start`

Operating System	Command/Task
OSX	`/opt/novell/usdrv/startup/usdrvd start`
Solaris	`/etc/init.d/usdrvd start`
Tru64	`/sbin/init.d/usdrvd start`
Windows	Use the Windows Services application to start the NetIQ Identity Manager Windows Script Driver service.

To stop the driver shim, perform the task appropriate for your operating system as shown in the following table:

Table 6-2 *Stopping the Driver Shim*

Operating System	Command/Task
AIX	`/etc/rc.d/init.d/usdrvd stop`
FreeBSD	`/usr/local/etc/rc.d/usdrvd stop`
HP-UX	`/sbin/init.d/usdrvd stop`
Linux	`/etc/init.d/usdrvd stop`
OSX	`/opt/novell/usdrv/startup/usdrvd stop`
Solaris	`/etc/init.d/usdrvd stop`
Tru64	`/etc/init.d/usdrvd stop`
Windows	Use the Windows Services application to stop the NetIQ Identity Manager Windows Script Driver service.

You can also run the driver shim on Windows from the command line by executing `wsdriver.exe` in the `bin` directory under the driver installation directory. Output is written to the console. Stop the driver shim by pressing Ctrl+Break. Running the driver shim this way is recommended only for development and testing.

Displaying Driver Shim Status

To see status and version information for the driver shim, use the appropriate command for your operating system as shown in the following table:

Table 6-3 *Displaying the Status of the Driver Shim*

Operating System	Command
AIX	`/etc/rc.d/init.d/usdrvd status`
FreeBSD	`/usr/local/etc/rc.d/usdrvd status`
HP-UX	`/sbin/init.d/usdrvd status`

Operating System	Command
Linux	`/etc/init.d/usdrvd status`
OSX	`/opt/novell/usdrv/startup/usdrvd status`
Solaris	`/etc/init.d/usdrvd status`
Tru64	`/etc/init.d/usdrvd status`
Windows	Use the Windows Services application.

Monitoring Driver Messages

The Scripting driver writes messages to the system log on Linux and UNIX, and the trace file on Windows (`trace.log` in the logs directory by default). For details about the messages written by the driver, see Appendix B, "System and Error Messages," on page 137.

7 Securing the Scripting Driver

The section describes best practices for securing the Identity Manager driver for Scripting. Topics include

For additional information about Identity Manager security, see the NetIQ® Identity Manager 3.6.1 Administration Guide on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).

Using SSL

Enable SSL for communication between the Metadirectory engine and the driver shim on the connected system. For more information, see "Use SSL" on page 26. If you don't enable SSL, you are sending information, including passwords, in the clear.

Physical Security

Keep your servers in a physically secure location with access by authorized personnel only.

Network Security

Require users outside of the corporate firewall to use a VPN to access corporate data.

Auditing

Track changes to sensitive information. Examine audit logs periodically. For details about using NetIQ Audit to monitor driver operation, see the NetIQ Audit Documentation Web site (http://www.novell.com/documentation/novellaudit20/index.html).

Driver Security Certificates

SSL uses security certificates to control, encrypt, and authenticate communications.

Ensure that the security certificate directory `/opt/novell/usdrv/keys` is appropriately protected on Linux or UNIX platforms and `C:\Novell\wsdrv\keys` is protected on Windows platforms. The installation program sets secure file permissions for these directories.

The Driver Shim and the Identity Manager engine communicate through SSL using a certificate created in the Identity Vault and retrieved by the Driver Shim during the installation process. For more information on this certificate and how to renew or install third-party certificates, refer to the *Identity Manager Administration Guide*.

The Embedded Remote Loader web interface uses a dynamically generated, self-signed certificate for SSL communication. The details of this certificate are as follows:

Subject: SSL Server

Issuer: SSL Server

Validity: 1 year

Serial Number: 0

Key: 1024-bit RSA

Renewal of this certificate automatically occurs when the Driver Shim is restarted on the connected platform.

Driver Shell Scripts

The driver uses scripts to perform updates on the connected system, and to collect changes made there. Ensure that the script directory is appropriately protected. The installation program sets secure file permissions for this directory where applicable.

The Change Log

The change log file contains information about events on the connected system, including passwords. It is encrypted, but it should be protected against access by unauthorized users. Ensure that the change log directory is appropriately protected. The installation program sets secure file permissions for this directory where applicable.

Driver Passwords

Use strong passwords for the Driver object and Remote Loader passwords, and restrict knowledge of them to authorized personnel. These passwords are stored in encrypted form in the security certificate directory keys. The installation program sets secure file permissions for this directory.

Driver Code

Ensure that the driver executable directory `bin` and the driver files in `/usr/sbin` (Linux/UNIX only) are appropriately protected. The installation program sets secure file permissions for these items where applicable.

Administrative Users

Ensure that accounts with elevated rights on the Metadirectory system, Identity Vault systems, and the connected systems are appropriately secure. Protect administrative user IDs with strong passwords.

Connected Systems

Ensure that connected systems can be trusted with account information, including passwords, for the portions of the tree that are configured as their base containers.

A Troubleshooting

This section provides information about troubleshooting the Identity Manager 4.8 driver for Scripting. Major topics include

Driver Status and Diagnostic Files

There are several log files that you can view to examine driver operation.

The System Log (Linux/UNIX only)

The system log is used by the Scripting driver shim to record urgent, informational, and debug messages. Examining these should be foremost in your troubleshooting efforts. For detailed message documentation, see Appendix B, "System and Error Messages," on page 137.

The location for the system log varies from system to system and is generally configured through /etc/syslog.conf. The amount of information that is logged by the driver can also be configured through this system log configuration file. The following is a sample fragment from /etc/syslog.conf:

```
# sample /etc/syslog.conf
#
*.err;kern.notice;auth.notice                      /dev/sysmsg
*.err;kern.debug;daemon.notice;mail.crit          /var/adm/messages

*.alert;kern.err;daemon.err                        operator
*.alert                                            root
```

The options in the first column determine which messages are logged. The options in the second column specify the destination file or user to send the log output to. For example, specifying *.err logs all messages with a priority of err or above. For more information about syslog priorities, view your system documentation using the `man syslog` command. Messages from the driver shim and messages from the scripts are logged with various priorities as shown in Table A-1. The information that is recorded depends on your syslog configuration.

Table A-1 Message Priorities

Message Topic	Priority
Script being called	DEBUG
Successful Linux or UNIX command execution	INFO
Publication events	INFO
Failures	ERR

The Trace File

The default trace file exists on the connected system as `trace.log` in the `logs` directory under the installation folder. A large amount of debug information can be written to this file. Use the trace level setting in your driver shim configuration file to control what is written to the file. For details about the driver shim configuration file, see "The Driver Shim Configuration File" on page 32.

Table A-2 Driver Shim Trace Levels

Trace Level	Description
0	No debugging
1–3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous level plus Remote Loader, driver, driver shim, and driver connection messages.
5–7	Previous level plus change log and loopback messages. Higher trace levels provide more detail.
8	Previous level plus driver status log, driver parameters, driver command line, driver security, driver Web server, driver schema, driver encryption, driver PAM, driver SOAP API, and driver include/exclude file messages.
9	Previous level plus low-level networking and operating system messages.
10	Previous level plus maximum low-level program details (all options).

The following is an example configuration line to set the trace level:

```
-trace 9
```

To view the trace file:

1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
2. Authenticate by using any username and the password that you specified as the Remote Loader password.
3. Click **Trace**.

The Script Output File

By default, script output is written to `script-trace.log` in the `logs` directory under the driver installation directory on the connected system. This file captures the output from all scripts executed by the driver shim. The location of the script output file is set in the driver parameters.

DSTrace

You can view Identity Manager information using the DSTrace facility on the Metadirectory server. Use Identity Console to set the tracing level. For example, trace level 2 shows Identity Vault events in XML documents, and trace level 5 shows the results of policy execution. Because a high volume of trace output is produced, we recommend that you capture the trace output to a file. For details about using DSTrace, see the *Identity Manager Administration Guide* on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).

The Status Log

The status log is a condensed summary of the events that have been recorded on the Subscriber and Publisher channels. This file exists on the connected system as `dirxml.log` in the logs directory under the driver installation directory. You can change the log level to specify what types of events to log. For details about using the status log, see the *Identity Manager Administration Guide* on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).

To view the status log:

1. Use a Web browser to access the driver shim at https://driver-address:8091. Substitute the DNS name or IP address of your driver for driver-address.
2. Authenticate by using any username and the password that you specified as the Remote Loader password.
3. Click **Status**.

Troubleshooting Common Problems

Driver Shim Installation Failure

- Ensure that you use the correct installation program for your operating system and that you are running on a supported operating system. For details, see Chapter 2, “Planning for the Scripting Driver,” on page 15.
- Ensure that you run the installation as `root` (Linux/UNIX) or Administrator (Windows) or equivalent.
- (Linux/UNIX only) Ensure that your package management software, such as RPM, is installed and up-to-date.

Driver Rules Installation Failure

Ensure that you use a version of Identity Console that supports your version of Identity Manager.

Driver Certificate Setup Failure

To set up certificates, the driver shim communicates with the Metadirectory server using the LDAP secure port (636).

- Ensure that eDirectory™ is running LDAP with SSL enabled. For details about configuring eDirectory, see the *NetIQ eDirectory Administration Guide*.
- Ensure that the connected system has network connectivity to the Metadirectory server.

You can use the following command to configure the certificate at any time:

On Linux or UNIX:

```
/opt/novell/usdrv/bin/usdrv -s
```

On Windows:

```
wsdriver -s
```

If you cannot configure SSL using LDAP, you can install the certificate manually:

1. Click **Certificate Management** > **Trusted Root Management** options from the Identity Console landing page. The **Trusted Root Container** check box will be selected by default. Select the **Trusted Root** check box
2. Select the appropriate trusted root certificate from the list and click the **export** icon.
3. In the next screen, do not select the check box for **Export Private key**.
4. Select **Base64** format, then click **Next**.
5. Use FTP or another method to store the file on the connected system as `ca.pem` in the `keys` directory under the driver installation directory.

Driver Start Failure

- Examine the status log and DSTrace output.
- The driver must be specified as a Remote Loader driver, even if the Identity Vault and connected system are the same computer. You can set this option in the Identity Console **Drivers** page.

- You must activate both Identity Manager and the driver within 90 days. The Driver Set Overview page in Identity Console shows when Identity Manager requires activation. The Driver Overview page shows when the driver requires activation.

  For details about activating NetIQ Identity Manager Products, see the *Identity Manager Installation Guide* on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).

For more information about troubleshooting Identity Manager engine errors, see the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).

Driver Shim Startup or Communication Failure

- Examine the trace file.
- Ensure that the connected system's operating system version is supported. For a list of supported operating systems, see Chapter 2, "Planning for the Scripting Driver," on page 15.
- Apply all patches for your operating system.
- Ensure that the Remote Loader and Driver object passwords that you specified while setting up the driver on the Metadirectory server match the passwords stored with the driver shim.

  To update these passwords on the connected system, use the `/opt/novell/usdrv/bin/usdrv -sp` (Linux/UNIX) or use the `wsdriver -sp` (Windows) command. The passwords are stored under keys in the driver installation directory in encrypted files `dpwdlf40` (Driver object password) and `lpwdlf40` (Remote Loader password).

  To update these passwords on the Metadirectory server, use Identity Console to update the driver configuration. For details, see "Driver Configuration Page" on page 27.
- Ensure that the correct host name and port number of the connected system are specified in the Driver Configuration Remote Loader connection parameters. You can change the port number (default 8090) in `usdrv.conf` (Linux/UNIX) or `wsdrv.conf` (Windows).

Users or Groups Are Not Provisioned to the Connected System

- Examine the status log, DSTRACE output, trace file, and script output file.
- To be provisioned, users and groups must be in the appropriate base container. You can view and change the base containers in Identity Console on the Global Config Values page of the **Drivers** module.
- To provision identities from the Identity Vault to the connected system, the driver Data Flow property must be set to Bidirectional or Identity Vault to Application. To change this value, re-import the driver rules file over your existing driver.
- The user that the driver is security equivalent to must have rights to read information from the base container. For details about the rights required, see Table 2-1 on page 18.

Users or Groups Are Not Provisioned to the Identity Vault

- Examine the status log, DSTRACE output, and trace file.
- Examine the User Base Container and Group Base Container GCV values. For more details, "Global Configuration Values Page" on page 29.

- To provision identities from the connected system to the Identity Vault, the driver Data Flow property must be set to Bidirectional or Application to Identity Vault. To change this value, reimport the driver rules file over your existing driver.
- The user that the driver is security equivalent to must have rights to update the base container. For details about the rights required, see Table 2-1 on page 18.

Identity Vault User Passwords Are Not Provisioned to the Connected System

- Examine the status log, DSTRACE output, and script output file.
- There are several password management properties available in Identity Console on the **Global Config Values** page of the **Drivers** module. Ensure that the connected system accepts passwords from the Identity Vault. To determine the right settings for your environment, view the help for the options, or see the *Identity Manager Administration Guide* on the Identity Manager 4.8 Documentation Web site (https://www.netiq.com/documentation/identity-manager-48/).
- Ensure that the user's container has an assigned Universal Password policy and that the Synchronize Distribution Password When Setting Universal Password GCV is set for this policy.

Connected System User Passwords Are Not Provisioned to the Identity Vault

- Examine the status log, DSTRACE output, and the trace file.
- There are several password management properties available in Identity Console on the **Global Config Values** page of the **Drivers** module. Ensure that at least one of the following options is set:
  - The Identity Vault Accepts Passwords from the Connected System
  - The Identity Vault Accepts Administrative Password Resets from the Connected System
  - To determine the right settings for your environment, view the help for the options, or see the *Identity Manager Administration Guide* on the Identity Manager 4.8 Documentation Web (https://www.netiq.com/documentation/identity-manager-48/).
  - If the Require Password Policy Validation before Publishing Passwords GCV is set, the user's password must satisfy the password rules in the password policy assigned to the user container.

Metadirectory Objects Are Not Modified, Deleted, Renamed, or Moved

- Examine the status log, DSTRACE output, trace file, and script output file.
- Examine the driver Data Flow setting to verify the authoritative source for identities.
- Identity Vault and connected system identities must be associated before events are synchronized. To view an identity's associations, use the **Object Inspector** page in Identity

Console and search for the object.

- Identity Vault move events can remove the identity from the base container monitored by the driver to a container that is not monitored by the driver. This makes the move appear to be a delete.

Shared Memory Errors (Linux/UNIX only)

Shared memory is used by the driver shim to safely and securely communicate with the scripts on Linux and UNIX. If the system shared memory segments become unusable, you must shut down the process and fix the shared memory segments.

Shared memory segments can become unusable on some UNIX systems if the driver shim is improperly terminated without detaching from the segments. For information about how to properly stop the driver shim, see "Starting and Stopping the Driver Shim" on page 121. You can use the ipcs system tool to locate these segments and the ipcrm tool to manually clear them as shown in the following example:

```
> ipcs -m
------ Shared Memory Segments --------
key        shmid    owner     perms     bytes    nattch    status
0x2a065bbd 1802241  root      600       16384    1
> ipcrm -m 1802241
```

The driver shim generates default segments of 16384 bytes with permissions at 600.

B System and Error Messages

Components of the Identity Manager 4.8 driver for Scripting write messages to the system log to report operational status and problems. For more information about the system log, see "The System Log (Linux/UNIX only)" on page 129 For detailed troubleshooting information, see Appendix A, "Troubleshooting," on page 129.

Each message begins with a code of 3-6 characters associated with the driver component that generated the message. Use this code to find message information quickly as follows:

- "CFG Messages" on page 137
- "CHGLOG Messages" on page 138
- "DOM Messages" on page 138
- "DRVCOM Messages" on page 139
- "HES Messages" on page 139
- "LWS Messages" on page 140
- "NET Messages" on page 141
- "NIX Messages (Linux/UNIX only)" on page 141
- "OAP Messages" on page 144
- "RDXML Messages" on page 145

CFG Messages

Messages beginning with CFG are issued by configuration file processing.

CFG001E Could not open configuration file filename.

Explanation:	Could not open the configuration file.
Possible Cause:	The file does not exist.
Possible Cause:	You don't have permission to read the file.
Action:	Ensure that the configuration file exists at the correct location and that you have file system rights to read it.

CFG002E Error parsing configuration file line:

Explanation:	The line is not formatted as a valid configuration statement and cannot be parsed.
Possible Cause:	The configuration file contains invalid or incorrect statements.
Action:	Correct the line in the configuration file.

CFG003W Configuration file line was ignored. No matching statement name found: <configline>.

Explanation: This line is formatted as a valid configuration file statement, but the statement is not recognized. The line is ignored.

Possible Cause: The statement is incorrectly typed or the statement name is used only in a newer version of the software.

Action: Correct the statement.

CFG004E Error parsing configuration file line. No statement name was found: <configLine>.

Explanation: Could not find a statement name on the configuration line.

Action: Correct the line in the configuration file to supply the required statement.

CFG005E A required statement *statement_id* is missing from the configuration file.

Explanation: The *statement_id* statement was not specified in the configuration file, but is required for the application to start.

Possible Cause: The configuration file is missing required statements.

Action: Add the required statement to the configuration file.

CHGLOG Messages

Messages beginning with CHGLOG are issued by change log processing.

CHGLOG000I nameversion Copyright 2005 Omnibond Systems, LLC. ID=code_id_string.

Explanation: This message identifies the system component version.

Action: No action is required.

DOM Messages

Messages beginning with DOM are issued by driver components as they communicate among themselves.

DOM0001W XML parser error encountered: errorString.

Explanation: An error was detected while parsing an XML document.

Possible Cause: The XML document was incomplete, or it was not a properly constructed XML document.

Action: See the error string for additional details about the error. Some errors, such as no element found, can occur during normal operation and indicate that an empty XML document was received.

DRVCOM Messages

Messages beginning with DRVCOM are issued by the include/exclude system.

DRVCOM000I nameversion Copyright 2005 Omnibond Systems, LLC. ID=code_id_string.

Explanation: This message identifies the system component version.

Action: No action is required.

DRVCOM001W Invalid include/exclude CLASS statement.

Explanation: The include/exclude configuration file contains an invalid CLASS statement.

Action: Correct the include/exclude configuration file with proper syntax.

DRVCOM002D An include/exclude Rule was added for class: class.

Explanation: The include/exclude configuration supplied a rule for the specified class.

Action: None.

DRVCOM003D An include/exclude Association Rule was added for association association.

Explanation: The include/exclude configuration supplied an association rule for the specified association.

Action: None.

HES Messages

Messages beginning with HES are issued by driver components as they use HTTP to communicate.

HES001E Unable to initialize the HTTP client.

Explanation: Communications in the client could not be initialized.

Possible Cause: Memory is exhausted.

Action: Increase the amount of memory available to the process.

HES002I Connecting to host host_name on port port_number.

Explanation: The client is connecting to the specified server.

Action: None.

HES003W SSL communications have an incorrect certificate. rc = rc.

Explanation: The security certificate for SSL services could not be verified.

Possible Cause: The certificate files might be missing or invalid.

Action: Obtain a new certificate.

LWS Messages

Messages beginning with LWS are issued by the integrated HTTP server.

LWS0001I Server has been initialized.

Explanation: The server has successfully completed its initialization phase.

Action: None. Informational only.

LWS0002I All services are now active.

Explanation: All of the services offered by the server are now active and ready for work.

Action: None. Informational only.

LWS0003I Server shut down successfully.

Explanation: The server processing completed normally. The server ends with a return code of `0`.

Action: No action is required.

LWS0004W Server shut down with warnings.

Explanation: The server processing completed normally with at least one warning. The server ends with a return code of 4.

Action: See the log for additional messages that describe the warning conditions.

LWS0005E Server shut down with errors.

Explanation: The server processing ended with one or more errors. The server ends with a return code of 8.

Action: See the log for additional messages that describe the error conditions.

LWS0006I Starting service.

Explanation: The server is starting the specified service.

Action: None. Informational only.

LWS0007E Failed to start service.

Explanation: The server attempted to start the specified service, but the service could not start. The server terminates processing.

Action: See the log for additional messages that describe the error condition.

LWS0008I Stopping all services.

Explanation: The server was requested to stop. All services are notified and will subsequently end processing.

Action: None. Informational only.

NET Messages

Messages beginning with NET are issued by driver components during verification of SSL certificates.

NET001W Certificate verification failed. Result is result.

Explanation: A valid security certificate could not be obtained from the connection client. Diagnostic information is given by result.

Possible Cause: A security certificate has not been obtained for the component.

Possible Cause: The security certificate has expired.

Possible Cause: The component certificate directory has been corrupted.

Action: Respond as indicated by result. Obtain a new certificate if appropriate.

NIX Messages (Linux/UNIX only)

Messages beginning with NIX are issued by the driver shim.

NIX000I nameversion Copyright 2005 Omnibond Systems, LLC. ID=code_id_string.

Explanation: This message identifies the system component version.

Action: No action is required.

NIX001S An error occurred attempting to attach the shared memory segment to an address space (errno=errno).

Explanation: The driver uses shared memory as the mechanism for providing information to the scripts. An error occurred attempting to attach the shared memory to a physical address for access.

Possible Cause: The calling process has no access permissions for the requested attach type.

Possible Cause: An invalid or non-page-aligned address was provided to the system routine.

Possible Cause: Memory could not be allocated for the descriptor or for the page tables.

Action: Restart the driver process and ensure that there are adequate memory resources. Verify that the driver process is run as root and has permissions to read its configuration files. Contact NetIQ® Support for additional instructions if necessary.

NIX002S An error occurred while attempting to allocate a shared memory segment (errno = errno).

Explanation: The driver uses shared memory as the mechanism for providing information to the scripts. An error occurred attempting to allocate a shared memory segment.

Possible Cause: The memory size was too small or too large.

Possible Cause: The system shared memory settings might not have adequate values

Possible Cause: The memory segment could not be created because it already exists. This could be caused by an abnormal termination of a previous driver process.

Possible Cause: All possible shared memory IDs have been taken.

Possible Cause: Allocating a segment of the requested size would cause the system to exceed the system-wide limit on shared memory.

Possible Cause: No shared memory segment exists for the given key.

Possible Cause: The user or process does not have permission to access the shared memory segment.

Possible Cause: No memory could be allocated for segment overhead..

Action: Restart the driver process and ensure that there is sufficient memory.

Action: Verify that the driver process is run as root and has permissions to read its configuration files.

Action: Verify that the driver process is run as root and has permissions to read its configuration files

Action: If there are other applications on the server that use shared memory, ensure that they are running, healthy, and do not conflict with the requirements for the driver.

Action: Contact NetIQ Support for additional instructions if necessary.

NIX003S An error occurred attempting to create a System V IPC key. The project identifier pathname = pathname.

Explanation: The driver uses shared memory as the mechanism for providing information to the scripts. An error occurred attempting to create the key used to specify the shared memory segment.

Possible Cause: The project pathname is invalid or does not exist.

Action: Restart the driver process.

Action: Ensure that the file pathname is correct and that the process has adequate permissions to read the path.

NIX004S An error occurred while writing data to shared memory (bytes = bytes, allocationSize = allocationSize).

Explanation:	The driver uses shared memory as the mechanism for providing information to the shell scripts. An error occurred while writing data from the driver process into the shared memory segment.
Possible Cause:	Invalid memory resources or internal error.
Action:	Contact NetIQ Support.

NIX005S An error occurred attempting to set an environment variable.

Explanation:	The driver uses environment variables for some of the communication between the driver and other processes called from the scripts. An error occurred setting an environment variable.
Possible Cause:	There was not enough space to allocate the new environment.
Action:	Restart the driver and ensure that there are adequate memory resources for the driver process.

NIX006S An error occurred attempting to execute the script [script].

Explanation:	The driver uses scripts to update the system for events from the Identity Vault. An error occurred while attempting to execute one of these scripts.
Possible Cause:	The script does not exist on the local system.
Possible Cause:	A memory or environment allocation failure occurred.
Action:	Restart the driver and ensure that the script exists on the local system.

NIX007S An error occurred attempting to terminate the script [script].

Explanation:	The driver uses scripts to update the system for events from the Identity Vault. An error occurred while attempting to terminate the script.
Possible Cause:	The script does not exist on the local system.
Possible Cause:	A memory or environment allocation failure occurred.
Action:	Restart the driver and ensure that the script exists on the local system.

NIX008S The shared memory tool was unable to retrieve a key from the environment.

Explanation:	The shared memory tool uses an environment variable to retrieve the key used to unlock the shared memory region and access driver shim data. The tool could not obtain the key from the environment.
Possible Cause:	The driver shim cannot set environment variables, or the environment has become corrupt during event processing.
Action:	Restart the driver shim process and clear any residual shared memory segments.Action:

OAP Messages

Messages beginning with OAP are issued by driver components while communicating among themselves.

OAP001E Error in SSL configuration. Verify system entropy.

Explanation:	Entropy could not be obtained for SSL.
Possible Cause:	A source of entropy is not configured for the system.
Action:	Obtain and configure a source of entropy for the system.

OAP002E Error in SSL connect. Network address does not match certificate.

Explanation:	The SSL client could not trust the SSL server it connected to, because the address of the server did not match the DNS name or IP address that was found in the certificate for the server
Possible Cause:	The appropriate credentials are missing from the configuration.
Action:	If you cannot resolve the error, collect diagnostic information and call NetIQ Support.

OAP003E Error in SSL connect. Verify address and port.

Explanation:	A TCP/IP connection could not be made.
Possible Cause:	The server is not running.
Possible Cause:	The configuration information does not specify the correct network address or port number.
Action:	Verify that the server is running properly.

OAP004E HTTP Error: cause.

Explanation:	The username or password provided failed basic authentication.
Possible Cause:	The username or password is incorrect.
Action:	Verify that username is in full context (cn=user,ou=ctx,o=org or user.ctx.org) and that the password was correctly typed.

OAP005E HTTP Error: Internal Server Error.

Explanation:	The server experienced an internal error that prevents the request from being processed.
Possible Cause:	A secure LDAP server is not available.
Action:	Ensure that the LDAP server is available.
Action:	Ensure that the LDAP host and port are configured correctly.

RDXML Messages

Messages beginning with RDXML are issued by the embedded Remote Loader.

RDXML000I nameversion Copyright 2005 Omnibond Systems, LLC. ID=code_id_string.

Explanation: This message identifies the system component version.

Action: No action is required.

RDXML001I Client connection established.

Explanation: A client has connected to the driver. This can be the Metadirectory engine connecting to process events to and from the driver, or a Web-based request to view information or publish changes through the SOAP mechanism.

Action: No action required.

RDXML002I Request issued to start Driver Shim.

Explanation: The driver received a command to start the driver shim and begin processing events.

Action: No action required.

RDXML003E An unrecognized command was issued. The driver shim is shutting down.

Explanation: The driver received an unrecognized command from the Metadirectory engine. The driver shim is shutting down to avoid further errors.

Possible Cause: Network error.

Possible Cause: Invalid data sent to the driver.

Possible Cause: The Metadirectory engine version might have been updated with new commands that are unrecognized by this version of the driver.

Possible Cause: This message is logged when the driver shim process is shut down from the connected system rather than from a Driver object request. The local system can queue an invalid command to the driver shim to simulate a shutdown request and terminate the running process.

Action: Ensure that the network connection is secured and working properly.

Action: Apply updates for the engine or driver if necessary.

Action: If the driver shim process was shut down from the local system, no action is required.

RDXML004I Client Disconnected.

Explanation: A client has disconnected from the driver. This might be the Metadirectory engine disconnecting after a driver shutdown request or a Web-based request that has ended.

Action: No action required.

RDXML005W Unable to establish client connection.

Explanation: A client attempted to connect to the driver, but was disconnected prematurely.

Possible Cause: The client is not running in SSL mode.

Possible Cause: Mismatched SSL versions or mismatched certificate authorities.

Possible Cause: Problems initializing SSL libraries because of improperly configured system entropy settings.

Action: Ensure that both the Metadirectory engine and the driver are running in the same mode: either clear text mode or SSL mode.

Action: If you are using SSL, ensure that the driver and Metadirectory engine have properly configured certificates, and that the driver system is configured properly for entropy.

RDXML006E Error in Remote Loader Handshake.

Explanation: The Metadirectory engine attempted to connect to the driver, but the authorization process failed. Authorization requires that both supply mutually acceptable passwords. Passwords are configured at installation.

Possible Cause: The Remote Loader or Driver object passwords do not match.

Action: Set the Remote Loader and Driver object passwords to the same value for both the driver and the driver shim. Use Identity Console to modify the driver properties. Re-configure the driver shim on the connected system.

RDXML007I Driver Shim has successfully started and is ready to process events.

Explanation: The Metadirectory engine has requested the driver to start the shim for event processing, and the driver shim has successfully started.

Action: No action required.

RDXML008W Unable to establish client connection from remoteName.

Explanation: A client attempted to connect to the driver, but was disconnected prematurely.

Possible Cause: The client is not running in SSL mode.

Possible Cause: Mismatched SSL versions or mismatched certificate authorities.

Possible Cause: Problems initializing SSL libraries because of improperly configured system entropy settings.

Action: Ensure that both the Metadirectory engine and the driver are running in the same mode: either clear text mode or SSL mode.

Action: If you are using SSL, ensure that the driver and Metadirectory engine have properly configured certificates, and that the driver system is configured properly for entropy.

RDXML009I Client connection established from remoteName.

Explanation: A client has connected to the driver. This can be the Metadirectory engine connecting to process events to and from the driver, or a Web-based request to view information or publish changes through the SOAP mechanism.

Action: No action required.

C IDMLib Reference

The Identity Manager 4.8 driver for Scripting provides an API library for accessing and updating data to and from the driver shim during event subscription and publication.

Major topics in this section include

UNIX Shell (idmlib.sh) Reference

The scripts are written for the Linux and UNIX Bourne Shell. They are located in the scripts folder below the folder where the driver was installed (`/opt/novell/usdrv/` by default).

Subscriber events are submitted to subscriber.sh, which then calls the script for the event. Modify the shell script file corresponding to the event type: `add.sh`, `modify.sh`, `modify-password.sh`, `delete.sh`, `move.sh`, `rename.sh`. Queries of the external system should be handled in query.sh.

The Publisher calls `poll.sh` periodically. The frequency of the poll is determined by the Polling Interval driver parameter (60 seconds by default). Edit `poll.sh` to allow the driver to respond to events in the external account management system.

The Publisher calls `heartbeat.sh` periodically to determine whether the external account management system is responding correctly.

The built-in functions below are defined in `idmlib.sh`.

General Functions

IDMGETDRVVAR ParamName

Returns the string value for the Driver parameter specified by the string ParamName.

TRACE Message

Appends the specified message to the user-defined trace file.

EXEC Command

Executes an external program using the specified command line, and returns its numerical exit code on completion.

STATUS Level Message

Sends a status document with given level and message to return to the Identity Manager engine when the script completes.

The status document as seen by the engine looks like the following:

```
<status level="success">This is a message</status>
```

STATUS_SUCCESS Message

Sends a status document with a success level and message to return to the Identity Manager engine when the script completes.

STATUS_WARNING Message

Sends a status document with a warning level and message to return to the Identity Manager engine when the script completes.

STATUS_RETRY Message

Sends a status document with a retry level and message to return to the Identity Manager engine when the script completes.

STATUS_ERROR Message

Sends a status document with a error level and message to return to the Identity Manager engine when the script completes.

STATUS_FATAL Message

Sends a status document with a fatal level and message to return to the Identity Manager engine when the script completes.

Subscriber Functions

- "IDMGETSUBVAR ParamName" on page 151
- "IDMGETVAR Name" on page 151
- "IDMSETVAR Name Value" on page 151

IDMGETSUBVAR ParamName

Returns the string value for the Subscriber parameter specified by the string ParamName.

IDMGETVAR Name

Returns a string value for the item specified by Name through standard output. If no values exist, Empty is returned. If the value is multi-valued, each value will be separated by a newline character.

IDMSETVAR Name Value

Sets a single string value for the item specified by Name to be returned to the driver engine.

Publisher Functions

Only one function exists in this category.

IDMGETPUBVAR ParamName

Returns the string value for the Publisher parameter specified by the string ParamName.

Query Functions

- "IDMQUERY ClassName Association ReadAttrs" on page 151
- "IDMGETQVAR ParamName" on page 151

IDMQUERY ClassName Association ReadAttrs

Performs a query to the engine with the given ClassName, Association and ReadAttrs

IDMGETQVAR ParamName

Retrieves a string value for the query result item, specified by ParamName, through standard output. If no values exist, Empty is returned. If the value is multi-valued, each value is separated by a newline character.

Heartbeat Functions

HEARTBEAT_SUCCESS Message

Use these functions in the `heartbeat.sh` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

HEARTBEAT_ERROR Message

Use these functions in the `heartbeat.sh` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

HEARTBEAT_WARNING Message

Use these functions in the `heartbeat.sh` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

Perl (IDMLib.pm) Reference

These scripts are written for the Linux and UNIX Perl interpreter. They are located in the scripts folder below the folder where the driver was installed (`/opt/novell/usdrv/` by default).

Subscriber events are submitted to `subscriber.pl`, which then calls the script for the event. Modify the Perl script file corresponding to the event type: `add.pl`, `modify.pl`, `modify-password.pl`, `delete.pl`, `move.pl`, `rename.pl`. Queries of the external system should be handled in `query.pl`.

The Publisher calls `poll.pl` periodically. The frequency of the poll is determined by the Polling Interval driver parameter (60 seconds by default). Edit `poll.pl` to allow the driver to respond to events in the external account management system.

The Publisher calls `heartbeat.pl` periodically to determine whether the external account management system is responding correctly.

The built-in functions below are defined in `IDMLib.pm`.

General Functions

idmgetvar($ParamName)

Returns the string value for the Driver parameter specified by the string ParamName.

idmtrace($Message)

Appends the specified message to the user-defined trace file.

exec($Command)

Executes an external program using the specified command line, and returns its numerical exit code on completion.

status($Level, $Message)

Sends a status document with given level and message to return to the Identity Manager engine when the script completes.

The status document as seen by the engine looks like the following:

```
<status level="success">This is a message</status>
```

status_success($Message)

Sends a status document with a success level and message to return to the Identity Manager engine when the script completes.

status_warning($Message)

Sends a status document with a warning level and message to return to the Identity Manager engine when the script completes.

status_retry($Message)

Sends a status document with a retry level and message to return to the Identity Manager engine when the script completes.

status_error($Message)

Sends a status document with a error level and message to return to the Identity Manager engine when the script completes.

status_fatal($Message)

Sends a status document with a fatal level and message to return to the Identity Manager engine when the script completes.

Subscriber Functions

- "idmgetsubvar($ParamName)" on page 154
- "idmgetvar($Name)" on page 154
- "idmsetvar($Name, $Value)" on page 154

idmgetsubvar($ParamName)

Returns the string value for the Subscriber parameter specified by the string ParamName.

idmgetvar($Name)

Returns a string value for the item specified by Name through standard output. If no values exist, Empty is returned. If the value is multi-valued, each value is separated by a newline character.

idmsetvar($Name, $Value)

Sets a single string value for the item specified by Name to be returned to the driver engine.

Publisher Functions

- "idmgetpubvar($ParamName)" on page 154

idmgetpubvar($ParamName)

Returns the string value for the Publisher parameter specified by the string ParamName.

Query Functions

- "idmquery($ClassName, $Association, $ReadAttrs)" on page 155
- "idmgetqva($ParamName)" on page 155

idmquery($ClassName, $Association, $ReadAttrs)

Performs a query to the engine with the given ClassName, Association and ReadAttrs.

idmgetqva($ParamName)

Retrieves a string value for the query result item, specified by ParamName, through standard output. If no values exist, Empty is returned. If the value is multi-valued, each value is separated by a newline character.

Heartbeat Functions

- "heartbeat_success($Message)" on page 155
- "heartbeat_error($Message)" on page 155
- "heartbeat_warning($Message)" on page 155

heartbeat_success($Message)

Use these functions in the `heartbeat.sh` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

heartbeat_error($Message)

Use these functions in the `heartbeat.sh` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

heartbeat_warning($Message)

Use these functions in the `heartbeat.sh` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

Python (idmlib.py) Reference

These scripts are written for the Linux and UNIX Python interpreter. They are located in the scripts folder below the folder where the driver was installed (`/opt/novell/usdrv/` by default).

Subscriber events are submitted to `subscriber.py,` which then calls the script for the event. Modify the Python script file corresponding to the event type: `add.py`, `modify.py`, `modify-password.py`, `delete.py`, `move.py`, `rename.py`. Queries of the external system should be handled in `query.py`.

The Publisher calls `poll.py` periodically. The frequency of the poll is determined by the Polling Interval driver parameter (60 seconds by default). Edit `poll.py` to allow the driver to respond to events in the external account management system.

The Publisher calls `heartbeat.py` periodically to determine whether the external account management system is responding correctly.

The built-in functions below are defined in `idmLib.py`.

General Functions

idmgetvar(VariableName)

Returns the string value for the Driver parameter specified by the string ParamName.

idmtrace(Message)

Appends the specified message to the user-defined trace file.

exec(Command)

Executes an external program using the specified command line, and returns its numerical exit code on completion.

status(Level, Message)

Sends a status document with given level and message to return to the Identity Manager engine when the script completes.

status_success(Message)

Sends a status document with a success level and message to return to the Identity Manager engine when the script completes.

status_warning(Message)

Sends a status document with a warning level and message to return to the Identity Manager engine when the script completes.

status_retry(Message)

Sends a status document with a retry level and message to return to the Identity Manager engine when the script completes.

status_error(Message)

Sends a status document with a error level and message to return to the Identity Manager engine when the script completes.

status_fatal(Message)

Sends a status document with a fatal level and message to return to the Identity Manager engine when the script completes.

Subscriber Functions

- "idmgetsubvar(VariableName)" on page 157
- "idmgetvar(Name)" on page 157
- "idmsetvar(Name, Value)" on page 157

idmgetsubvar(VariableName)

Returns the string value for the Subscriber parameter specified by the string VariableName.

idmgetvar(Name)

Returns a string value for the item specified by Name through standard output. If no values exist, Empty is returned. If the value is multi-valued, each value is separated by a newline character.

idmsetvar(Name, Value)

Sets a single string value for the item specified by Name to be returned to the driver engine.

Publisher Functions

- "idmgetpubvar(VariableName)" on page 157

idmgetpubvar(VariableName)

Returns the string value for the Publisher parameter specified by the string VariableName.

Query Functions

- "idmquery(ClassName, Association, ReadAttrs)" on page 158
- "idmgetqva(ParamName)" on page 158

idmquery(ClassName, Association, ReadAttrs)

Performs a query to the engine with the given ClassName, Association and ReadAttrs.

idmgetqva(ParamName)

Retrieves a string value for the query result item, specified by ParamName, through standard output. If no values exist, Empty is returned. If the value is multi-valued, each value is separated by a newline character.

Heartbeat Functions

- "heartbeat_success(Message)" on page 158
- "heartbeat_error(Message)" on page 158
- "heartbeat_warning($Message)" on page 158

heartbeat_success(Message)

Use these functions in the `heartbeat.py` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

heartbeat_error(Message)

Use these functions in the `heartbeat.py` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

heartbeat_warning($Message)

Use these functions in the `heartbeat.py` script to indicate the status of the external application. Heartbeat documents are sent to the engine in following format:

```
  <status level="success" type="heartbeat">This is a heartbeat message</
status>
```

Microsoft VBScript (IDMLib.vbs) Reference

The scripts are written using Microsoft VBScript. They are located in the scripts folder below the folder where the driver was installed (`C:\Program Files\Novell\WSDriver` by default).

Subscriber events are submitted to `Subscriber.wsf`, which then calls the script for the event. Modify the VBS file corresponding to the event type: `Add.vbs`, `Modify.vbs`, `ModifyPassword.vbs`, `Delete.vbs`, Move.vbs, `Rename.vbs`. Queries of the external system should be handled in `Query.vbs`.

The Publisher calls `Poll.wsf` periodically. The frequency of the poll is determined by the Polling Interval driver parameter (60 seconds by default). Edit `Poll.wsf` to allow the driver to respond to events in the external account management system.

The Publisher calls `Heartbeat.wsf` periodically to determine whether the external account management system is responding correctly.

Topics discussing the built-in functions in `IDMLib.vbs` are categorized as follows:

- "General Functions" on page 159
- "Subscriber Functions" on page 160
- "Publisher Functions" on page 162
- "Query Functions" on page 162
- "Heartbeat Functions" on page 165

General Functions

- "Function IDMGetDriverParam(ParamName)" on page 159
- "Sub IDMTrace Message" on page 159
- "Function IDMExecute(Command)" on page 160
- "Function IDMExecuteIO(Command, Input)" on page 160
- "Sub IDMStatus(Level, Message)" on page 160
- "Sub IDMStatusSuccess(Message)" on page 160
- "Sub IDMStatusWarning(Message)" on page 160
- "Sub IDMStatusRetry(Message)" on page 160
- "Sub IDMStatusError(Message)" on page 160
- "Sub IDMStatusFatal(Message)" on page 160

Function IDMGetDriverParam(ParamName)

Returns the string value for the Driver parameter specified by the string ParamName.

Sub IDMTrace Message

Appends the specified message to the user-defined trace file.

Function IDMExecute(Command)

Executes an external program using the specified command line, and returns its numerical exit code on completion.

Function IDMExecuteIO(Command, Input)

Executes an external program using the specified command line, submits the strings from array Input on standard input, and returns output from standard output and standard error as an array. You can specify Empty for the Input parameter. The function returns when the program completes. The first element of the returned array is the program exit code. Subsequent elements (if any) are strings, one for each line that was output to standard output and standard error.

Sub IDMStatus(Level, Message)

Set the status level and message to return to the Identity Manager engine when the script completes.

Sub IDMStatusSuccess(Message)

Set the status level and message to return to the Identity Manager engine when the script completes.

Sub IDMStatusWarning(Message)

Set the status level and message to return to the Identity Manager engine when the script completes.

Sub IDMStatusRetry(Message)

Set the status level and message to return to the Identity Manager engine when the script completes.

Sub IDMStatusError(Message)

Set the status level and message to return to the Identity Manager engine when the script completes.

Sub IDMStatusFatal(Message)

Set the status level and message to return to the Identity Manager engine when the script completes.

Subscriber Functions

- “Function IDMGetSubscriberParam(ParamName)” on page 161
- “Sub IDMSetCommand(Command)” on page 161

Function IDMGetSubscriberParam(ParamName)

Returns the string value for the Subscriber parameter specified by the string ParamName.

Sub IDMSetCommand(Command)

Sets the command that the Subscriber return to the Identity Manager engine. This function must be called before using IDMWriteValue functions. If only a status needs to be returned, use one of the IDMStatus functions (see above).

Function IDMGetEventValueCount(Name)

Returns the number of values for the item specified by Name. (Items include event information and attribute changes.)

Function IDMGetEventValues(Name)

Returns an array of string values for the item specified by Name. If no values exist, Empty is returned.

Function IDMGetEventValue(Name)

Returns the string value for the item specified by Name. If multiple values exist for the item, it returns the first value. If no values exist, Empty is returned.

Sub IDMWriteValue(Name, Value)

Sets a single string value for the item specified by Name to be returned to the driver engine when the script completes. You must call IDMSetCommand or one of the IDMStatus functions before calling this function.

Sub IDMWriteValues(Name, Values)

Sets an array of string values for the item specified by Name to be returned to the driver engine when the script completes. You must call IDMSetCommand or one of the IDMStatus functions before calling this function.

Function IDMSubGetNamedPassword(Name)

Returns a named password specifed by Name from the Identity Manager engine. The value Empty is returned if no such password exists.

Publisher Functions

Function IDMGetPublisherParam(ParamName)

Returns the string value for the Publisher parameter specified by the string ParamName.

Sub IDMPublishInit(Command)

Sets the Publisher command specified by Command to return to the driver engine when IDMPublish is called.

Sub IDMPublishValue(Name, Value)

Sets a single string value for the item specified by Name to be returned to the driver engine when IDMPublish is called.

Sub IDMPublishValues(Name, Values)

Sets an array of string values for the item specified by Name to be returned to the driver engine when IDMPublish is called.

Function IDMPublish

Submit the command and item values specified above to the driver engine for Publication to the identity vault.

Function IDMPubGetNamedPassword(Name)

Returns a named password specified by Name from the Identity Manager engine. The value Empty is returned if no such password exists.

Query Functions

Sub IDMQueryInit

Initializes a query to be submitted to the identity vault with the IDMQuery call. NOTE: Currently only queries that query a single object are supported.

Sub IDMQuerySetAssociation(Association)

Specifies the association of the identity vault object to query.

Sub IDMQuerySetSearchRoot(SearchRoot)

Specifies the DN of the identity vault object to query. Either the object's association or DN must be specified. If both are specified, the association value is used by the Identity Manager engine.

Sub IDMQueryAddSearchAttr(Name, Value)

Specifies a search condition to be used for the query, of the form Name=Value. Name specifies an attribute, and Value specifies a value it must match. The query will return only objects matching all specified conditions.

Sub IDMQueryAddReadAttr(Name)

Specifies an attribute name whose values should be returned by the query. By default, all attributes are returned.

Sub IDMQuerySetReadParent(ReadParent)

Specifies whether the association and DN of the parent of the queried object should be returned (ReadParent is boolean). The default is False.

Function IDMQuery

Executes the query with the parameters specified by IDMQuerySetXXX calls. The function returns True if an object (called an instance) is returned.

Function IDMGetQueryInstanceAssociation

Returns the association for the returned instance.

Function IDMGetQueryInstanceDN

Returns the DN for the returned instance. The DN is in slash format, such as \ACME\Users\Bob.

Function IDMGetQueryInstanceClass

Returns the class name for the returned instance.

Function IDMGetQueryInstanceParentAssociation

Returns the association for instance's parent object, if the ReadParent flag was specified.

Function IDMGetQueryInstanceParentDN

Returns the DN for instance's parent object, if the ReadParent flag was specified.

Function IDMGetQueryInstanceAttrNames

Returns an array containing the names of the attributes retrieved for the instance. Returns Empty if no attributes were retrieved.

Function IDMGetQueryInstanceAttrCount

Returns the number of attributes retrieved for the instance.

Function IDMGetQueryInstanceAttrValues(AttrName)

Returns an array of values for the attribute with the specified AttrName. Returns Empty if no values are available.

Function IDMGetQueryInstanceAttrValue(AttrName)

Returns a string value for the attribute with the specified AttrName. If multiple values are available for the attribute, the first one is returned. If no values are available, Empty is returned.

Heartbeat Functions

- "Sub IDMHeartbeatSuccess(Message)" on page 165
- "Sub IDMHeartbeatError(Message)" on page 165
- "Sub IDMHeartbeatWarning(Message)" on page 165

Sub IDMHeartbeatSuccess(Message)

Use these functions in the heartbeat.wsf script to indicate the status of the external application.

Sub IDMHeartbeatError(Message)

Use these functions in the `heartbeat.wsf` script to indicate the status of the external application.

Sub IDMHeartbeatWarning(Message)

Use these functions in the `heartbeat.wsf` script to indicate the status of the external application.

Windows PowerShell (IDMLib.ps1) Reference

The scripts are written using Windows PowerShell. They are located in the `scripts\powershell` folder below the folder where the driver was installed (`C:\Program Files\Novell\WSDriver` by default).

Subscriber events are submitted to `Subscriber.ps1`, which then calls the script for the event. Modify the ps1 file corresponding to the event type: `Add.ps1`, `Modify.ps1`, `ModifyPassword.ps1`, `Delete.ps1`, Move.ps1, `Rename.ps1`. Queries of the external system should be handled in `Query.ps1`.

The Publisher calls `Poll.ps1` periodically. The frequency of the poll is determined by the Polling Interval driver parameter (60 seconds by default). Edit `Poll.ps1` to allow the driver to respond to events in the external account management system.

The Publisher calls `Heartbeat.ps1` periodically to determine whether the external account management system is responding correctly.

Topics discussing the built-in functions in `IDMLib.ps1` are categorized as follows:

- "General Functions" on page 165
- "Subscriber Functions" on page 166
- "Publisher Functions" on page 168
- "Query Functions" on page 169
- "Heartbeat Functions" on page 171

General Functions

- "function idm_getdriverparam($paramname)" on page 166
- "function idm_trace($message)" on page 166

function idm_getdriverparam($paramname)

Returns the string value for the Driver parameter specified by the string $paramname.

function idm_trace($message)

Appends the specified message to the user-defined trace file.

function idm_status($level, $message)

Set the status level and message to return to the Identity Manager engine when the script completes.

function idm_statussuccess($message)

Set the status success message to return to the Identity Manager engine when the script completes.

function idm_statuswarning($message)

Set the status warning message to return to the Identity Manager engine when the script completes.

function idm_statusretry($message)

Set the status retry message to return to the Identity Manager engine when the script completes.

function idm_statuserror($message)

Set the status error message to return to the Identity Manager engine when the script completes.

function idm_statusfatal($message)

Set the status fatal message to return to the Identity Manager engine when the script completes.

Subscriber Functions

function idm_getsubscriberparam($paramname)

Returns the string value for the Subscriber parameter specified by the string $paramname.

function idm_setcommand($command)

Sets the command that the Subscriber returns to the Identity Manager engine. This function must be called before using idm_writevalue functions. If only a status needs to be returned, use one of the idm_status functions (see above).

function idm_geteventvalues($name)

Returns an array of string values for the item specified by $name. If no values exist, $null is returned.

function idm_geteventvalue($name)

Returns the string value for the item specified by $name. If no values exist, $null is returned.

function idm_geteventvaluenames

Returns an array containing each value name for the event. This function can be used to iterate over every value.

function idm_geteventattrnames

Returns an array containing each attribute item for the event. This includes ADD_*attrname*, REMOVE_*attrname* and PASSWORD values.

function idm_writevalues($name, $values)

Sets an array of string values for the item specified by $name to be returned to the driver engine when the script completes. You must call idm_setcommand or one of the idm_status functions before calling this function.

function idm_writevalue($name, $value)

Sets a single string value for the item specified by $name to be returned to the driver engine when the script completes. You must call idm_setcommand or one of the idm_status functions before calling this function.

function idm_subgetnamedpassword($name)

Returns a named password specifed by $name from the Identity Manager engine. The value $null is returned if no such password exists.

Publisher Functions

function idm_getpublisherparam($paramname)

Returns the string value for the Publisher parameter specified by the string $paramname.

function idm_publishinit($command)

Sets the Publisher command specified by $command to return to the driver engine when idm_publish is called.

function idmpublishvalues($name, $values)

Sets an array of string values for the item specified by $name to be returned to the driver engine when idm_publish is called.

function idm_publishvalue($name, $value)

Sets a single string values for the item specified by $name to be returned to the driver engine when idm_publish is called.

function idm_publish

Submit the command and item values specified above to the driver engine for Publication to the identity vault.

function idm_pubgetnamedpassword($name)

Returns a named password specified by $name from the Identity Manager engine. The value $null is returned if no such password exists.

Query Functions

function idm_queryinit

Initializes a query to be submitted to the identity vault with the idm_doquery call. NOTE: Currently only queries that query a single object are supported.

function idm_querysetassociation($association)

Specifies the association of the identity vault object to query.

function idm_querysetsearchroot($searchroot)

Specifies the DN of the identity vault object to query. Either the object's association or DN must be specified. If both are specified, the association value is used by the Identity Manager engine.

function idm_queryaddsearchattr($name, $value)

Specifies a search condition to be used for the query, of the form $name=$value. $name specifies an attribute, and $value specifies a value it must match. The query will return only objects matching all specified conditions.

function idm_queryaddreadattr($name)

Specifies an attribute name whose values should be returned by the query. By default, all attributes are returned.

function idm_querysetreadparent($readparent)

Specifies whether the association and DN of the parent of the queried object should be returned ($readparent is boolean). The default is $False.

function idm_doquery

Executes the query with the parameters specified by idm_querysetXXX calls. The function returns $True if an object (called an instance) is returned.

function idm_getqueryinstanceassociation

Returns the association for the returned instance.

function idm_getqueryinstancedn

Returns the DN for the returned instance. The DN is in slash format, for example: `\ACME\Users\Bob`.

function idm_getqueryinstanceclass

Returns the class name for the returned instance.

function idm_getqueryinstanceparentassociation

Returns the association for instance’s parent object, if the $readparent flag was specified.

function idm_getqueryinstanceparentDN

Returns the DN for instance’s parent object, if the $readparent flag was specified.

function idm_getqueryinstanceattrnames

Returns an array containing the names of the attributes retrieved for the instance. Returns $null if no attributes were retrieved.

function idm_getqueryinstanceattrcount

Returns the number of attributes retrieved for the instance.

function idm_getqueryinstanceattrvalues($attrname)

Returns an array of values for the attribute with the specified $attrname. Returns $null if no values are available.

function idm_getqueryinstanceattrvalue($attrname)

Returns a string value for the attribute with the specified $attrname. If multiple values are available for the attribute, the first one is returned. If no values are available, $null is returned.

Heartbeat Functions

- "function idmheartbeatsuccess($message)" on page 171
- "function idmheartbeaterror($message)" on page 171
- "function idmheartbeatwarning($message)" on page 171

function idmheartbeatsuccess($message)

Use this function in the `heartbeat.ps1` script to indicate a success status of the external application.

function idmheartbeaterror($message)

Use this function in the `heartbeat.ps1` script to indicate an error status of the external application.

function idmheartbeatwarning($message)

Use this function in the `heartbeat.ps1` script to indicate a warning status of the external application.

D Technical Details

Topics in this section include

Using the usdrv-config Command (Linux/UNIX only)

You can use `/usr/sbin/usdrv-config` to change the driver shim configuration. When you run this command, you are prompted for the function to perform.

```
> usdrv-config
Which configuration do you want to perform?
1) Set the Remote Loader and Driver object passwords
2) Configure the driver for Secure Sockets Layer (SSL)
Select one configuration option [q/?]:
```

Enter the number of the function you want to configure, then respond to the prompts.

Setting the Remote Loader and Driver Object Passwords

The `usdrv-config` command prompts you to enter and confirm the Remote Loader password and the Driver object password.

```
Enter Remote Loader password:
Confirm Remote Loader password:
Enter Driver object password:
Confirm Driver object password:
```

The Remote Loader password is used by the Metadirectory engine to authenticate itself to the driver shim (embedded Remote Loader). The Driver object password is used by the driver shim to authenticate itself to the Metadirectory engine.

The Remote Loader and Driver object passwords set by usdrv-config are stored on the connected system. The Remote Loader and Driver object passwords set for the driver using Identity Console are stored in the Identity Vault. Each password on the connected system must exactly match its counterpart in the Identity vault.

To change the passwords after driver installation:

1. In Identity Console, click the Identity Manager **Drivers** module from the landing page.
2. Click the specific driver to edit.
3. Select the **Connection Parameters** tab.
4. Ensure **Connect to Remote Loader** is selected.

5 Specify the Driver object password.

6 Specify the Remote Loader password.

The Remote Loader password is below the Authentication heading.

7 Click **Apply**.

8 Restart the driver.

Configuring the Driver for SSL

The `usdrv-config` command prompts you to enter the LDAP server host address and port, then displays the Certificate Authority for that server and asks you if you accept it.

```
You are about to connect to the eDirectory LDAP server to retrieve
the eDirectory Tree Trusted Root public certificate.

Enter the LDAP Server Host Address [localhost]: sr.digitalairlines.com
Enter the LDAP Server Port [636]:

Certificate Authority:
   Subject:       ou=Organizational CA,o=TREENAME
   Not Before:    20070321144845Z
   Not After:     20170321144845Z
Do you accept the Certificate Authority? (Y/N) y
```

Enter the host name or IP address and TCP port number of an LDAP server for your Identity Vault. The LDAP server must be configured for SSL, and it must be listening on the SSL port. The default SSL port is 636.

The driver shim connects to the specified server and displays information about the Certificate Authority. If you accept the Certificate Authority, the driver shim saves it to the local file system.

If you do not have LDAP configured for SSL, you can use a manual process to configure the driver for SSL. For details, see Section A.2.3, "Driver Certificate Setup Failure."

Driver Shim Command Line Options

The following options can be specified on the driver shim (usdrv on Linux and UNIX, wsdriver on Windows) command line. You can also specify driver shim command line options as driver shim configuration file statements. For details about the driver shim configuration file, see “The Driver Shim Configuration File” on page 32.

Options Used to Set Up Driver Shim SSL Certificates

The following command line options are used to set up the driver shim SSL certificates:

Table D-1 Driver Shim Command Line Options for Setting Up SSL Certificates

Option (Short and Long Forms)	Description
-s -secure	Secures the driver by creating SSL certificates, then exits.
-p -password	Specifies the Remote Loader password

Other Options

Table D-2 Other Driver Shim Command Line Options

Option (Short and Long Forms)	Description
-c <congFile> -config <configFile>	Instructs the driver shim to read options from the specified configuration file. Options are read from `conf/usdrv.conf` (Linux/UNIX) or `conf\wsdrv.conf` (Windows) by default.
-sp [remoteLoaderPassword driverObjectPassword] -setpassword [remoteLoaderPassword driverObjectPassword]	Sets the Remote Loader and driver object passwords to the passwords specified on the command line, then stops the driver shim. If the passwords are omitted, the driver shim prompts for the passwords.
-installService	Creates a Windows service for the driver shim called NetIQ Identity Manager Windows Script Driver (Windows only).
-removeService	Removes the Windows service NetIQ Identity Manager Windows Script Driver (Windows only).
-? -help	Displays the command line options, then exits.
-v-version	Displays the driver shim version and build date, then exits.

Publisher Change Log Tool

The publisher channel might submit events to be published, using the change log tool usclh (on UNIX) or idmevent.exe (on Windows). These tools will create an event, which will be picked up by the driver shim on a polling interval and published to the Identity Manager engine, where it can be processed by Policy. The change log tool can be invoked at anytime on the application system. One commonly-used technique is to call the changelog tool from the polling script, which is executed on the polling interval as well. In such a scenario, the polling script can determine what changed and submit the changes to the change log to be processed immediately after the polling script terminates. However, if you wish to invoke the change log tool from another mechanism, events will be queued in the changelog and published on intervals when necessary.

The syntax for the change log tool on UNIX, usclh, is as follows:

```
usclh -t <type>
        [-c class]
        [-e event-id]
        [-a association]
        [-s src-dn]
        [-o old-src-dn]
        [-p password]
        [-w old-password]
        [-n new-name]
        [-r]
        [-y old-association]
        [-z new-association]
        [-l status-level]
        [-m status-message]
        [-1 | -2]
        [-?]
```

Where each option is described in the following table:

Table D-3 *Options*

Name	**Description**
type	The command type, which could be one of the following: `add`, `delete`, `modify`, `modify-password`, `rename`, `modify-association`, `status`, `xds`. When using the `xds` type, a raw XML document can be passed to the tool to be published as is.
event-id	The event-id of the document to be published. Typically, this is a timestamp or a counter. If none is specified, a default timestamp will be used.
association	The association string value for which the event being published describes.
src-dn	The source distinguished name of the object being published, if this object resides in a hierarchical directory structure.
old-src-dn	If the published event is a move or rename, the old-src-dn specifies the old distinguished name before the move or rename event.
password	The new password of the object being published. This is only valid for add or modify-password events.
old-password	The old password of the object being published. This is only valid for modify-password events.
new-name	The new name of an object being published during a rename event.
-r	If specified, instructs the event to remove the old name during a rename event.

Name	Description
old-association	Specifies the name of the old association value, during a modify-association event.
new-association	Specifies the name of the new association value, during a modify-association event.
status-level	Specifies the status level for a status message. Valid levels are: success, error, warning, retry, fatal.
status-message	Specifies the text messages that should be included for a status document.
-1	Specifies that the event should be put on hold (do not publish), until a release is issued.
-2	Specifies that all events on hold should be released (publishable).
-?	This help menu.

When invoked, the changelog utility waits for input on standard input until an EOF (end of file) character is received. If entered on the command-line, you can terminate it with the Ctl-d meta character. Additional name/value pairs can then be passed to this tool to supply additional event information such as attribute values being added or removed.

When invoked from a script, you can use a “here-is” document format to pass standard input to the changelog tool. When passing input to a command-line utility through standard input, you have the advantage that the information is protected from the environment, adding security to your publisher. When using command-line arguments, these options will appear in cleartext to the outside environment with tools such as “ps”.

Examples from a script:

```
usclh -t add -c User -a bob <<EOF
ADD_CN=bob
ADD_Login Disabled=true
EOF

usclh -t modify -c User -a bob <<EOF
ADD_CN=bob
ADD_Login Disabled=true
EOF

usclh -t modify-password -c User -a bob <<EOF
OLD_PASSWORD=secret
PASSWORD=newsecret
EOF

usclh -t rename -c User -a bob -n bob2 -r <<EOF

EOF
```

Examples from a command line:

```
usclh -t add -c User -a bob
ADD_CN=bob
ADD_Login Disabled=true
^d

usclh -t delete -c User -a bob
^d

usclh -t modify-password -c User -a bob -w secret -p newsecret
^d

# ./usclh -t xds
<nds dtdversion="1.1" ndsversion="8.6">
<input>
<modify class-name="User" event-id="12345">
  <association>bob</association>
  <modify-attr attr-name="MyAttr">
    <remove-all-values/>
    <add-value>
      <value>some new value</value>
    </add-value>
  </modify-attr>
</modify>
</input>
</nds>
^d
```

Files and Directories Modified by Installing the Driver Shim

Driver Shim Directory

When you install the driver, the `/opt/novell/usdrv` or `C:\Program Files\Novell\WSDriver` directory is created and populated with driver-related files and subdirectories.

/usr/sbin Files (Linux/UNIX only)

The following commands are added to `/usr/sbin`:

Table D-4 Driver Commands Placed in /usr/sbin

Command	Function
`usdrv-uninstall`	Uninstalls the Scripting driver
`usdrv-config`	Updates the configuration

init.d Files (Linux/UNIX only)

Commands to start, stop, and display the status of the driver are added to the appropriate file for the connected system operating system.

Table D-5 Commands for Starting, Stopping, and Displaying the Status of the Driver Shim

Operating System	Command
AIX	`/etc/rc.d/init.d/usdrvd`
HP-UX	`/sbin/init.d/usdrvd`
Linux	`/etc/init.d/usdrvd`
Solaris	`/etc/init.d/usdrvd`

Man Pages (Linux/UNIX only)

The installation process adds man pages for the driver shim, change log update command, and shared memory tool to `/usr/man`.

Driver Shim Configuration File

The installation program places a default driver shim configuration file at `/etc/usdrv.conf` on Linux and UNIX. On Windows, this file is `wsdrv.conf` in the conf directory in the installation directory.

Windows Support Files (Windows only)

Support files such as DLLs for the Visual C++ runtime are installed on Windows systems in the WinSxS directory (usually `C:\Windows\WinSxS`). If the driver shim is uninstalled, these files are removed.