Novell^tm

DirXML® Driver 2.0 for

Java* Message Service (JMS)

and WebSphere* MQ

Driver Overview and Configuration

Version 2.0

This document contains trade secrets and proprietary information. No use or disclosure of the information contained herein is permitted without the prior written consent.

All trade names, trademarks, or registered trademarks are trade names, trademarks, or registered trademarks of their respective companies.

Table of Contents

Preface

This document is for network administrators, consultants, and JMS administrators.

DirXML™ Driver 2.0 for Java* Message Service (JMS) and WebSphere* MQ is designed to share data between eDirectory™ and applications that are interfaced to the many different messaging bus applications that are supported by the JMS standard.

This configurable solution gives organizations the ability to increase productivity and streamline business processes by integrating any business process and information system platform in the enterprise with Directory Services.

The previous version of the driver is the DirXML Driver 1.0 for WebSphere MQ.

Documentation Conventions

The term driver refers to all components of the DirXML Driver 2.0 for Java Message Service (JMS) and WebSphere MQ and not to any one particular component.

In Novell documentation, a greater than symbol (>) is used to separate actions within a step and items in a cross-reference path.

In this documentation, a trademark symbol (®,™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

Introducing the Driver

Overview

The DirXML® Driver for Java* Message Service (JMS) and WebSphere* MQ, subsequently referred to as the driver or JMS driver, allows the creation of automated data synchronization links between any applications supported by a JMS-compliant message bus and Novell® eDirectory.

The JMS driver offers several key advantages for Secure Identity Management (SIM) integration, including the following:

- Native XML-based API connectivity to Enterprise Resource Planning applications from SAP, Oracle, JD Edwards and many others. This is often the only method authorized by customers and application vendors for interfacing with this class of application, as the XML API's can be pre-validated prior to processing. Hundreds of XML API's have been defined collaboratively by vendors and customers through The Organization for the Advancement of Structured Information Standards (OASIS) (http://www.oasis-open.org/).

  In addition to protecting data integrity, an XML-based API is highly abstracted (i.e. much less likely to change than the underlying table structure), which means that database modifications or even a wholesale application replacement will not necessarily require that the DirXML solution be modified in any way.

- Native XML-based connectivity to and from web application servers including IBM Websphere, BEA WebLogic, iPlanet AS, Oracle AS, Novell exteNd and others;

- Simplified integration into IBM Mainframe (OS/390) and Midrange (AS/400) environments as well as integration with security frameworks such as IBM RACF.
- Integration with the vast majority of Enterprise Application Integration (EAI) architectures, including TIBCO Rendezvous, IBM's WebsphereMQ and Websphere Business Integration (CrossWorlds), Oracle Advanced Queuing, Novell jBroker and many other such technologies.

The driver also provides Enterprise-class features that are needed in mission-critical situations:

- **Automated Fail-over**: multiple JMS driver instances work cooperatively to ensure high availability.

- **Load Balancing**: any number of JMS driver instances can be clustered to provide very high performance.

  In short, this DirXML driver enables identity profile information managed within business processes across the enterprise to be integrated into a common directory services strategy. It works with, rather than in place of, existing customer EAI infrastructures; it allows the guaranteed delivery, exchange and processing of Identity information among applications and eDirectory; and can greatly reduce the complexity and total number of DirXML drivers needed to implement an enterprise SIM solution.

What is JMS

The Java Message Service (JMS) API is an API for accessing enterprise messaging systems. It is part of the Java 2 Platform, Enterprise Edition (J2EE). This makes it easy to write business applications that asynchronously send and receive critical business data and events.

JMS defines a common enterprise messaging API that is designed to be easily and efficiently supported by a wide range of enterprise messaging products it supports both message queuing and publish-subscribe styles of messaging.

Java Message Service (JMS): http://java.sun.com/products/jms

Java 2 Platform, Enterprise Edition (J2EE): http://java.sun.com/j2ee

JMS Frequently Asked Questions: http://java.sun.com/products/jms/faq.html

Additional Information on JMS: http://java.sun.com/products/jms/docs.html

Understanding JMS Driver Concepts

The driver is a bi-directional synchronization product that establishes a link between JMS-supported systems and eDirectory. This solution uses XML to provide data and event transformation capabilities and can convert eDirectory data and events into XML data and vice-versa.

The JMS-compliant messaging queue (MQ) middleware generally acts as a hub that applications and directories can send and receive data to. The driver acts as a specialized connector into this hub that exchanges data among consumer applications and foreign directories and eDirectory. This results in two main flows of data for Identity Manager: the Publisher channel and the Application Subscriber channel.

The driver supports both Point-to-Point and Publish-Subscribe messaging models. Applications can publish information in any available form to queues and topics. The Publisher channel can also consume messages from queues or topics.

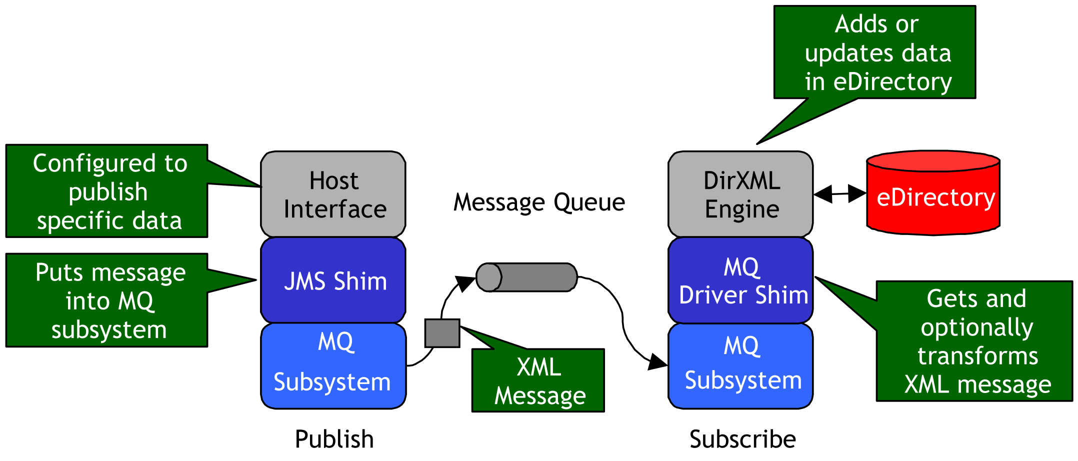
If the message queue system supports publish and subscribe type transactions, multiple consuming applications can listen for and receive the same XML documents. This type of message configuration has the ability to simplify or eliminate unnecessary DirXML application development by allowing more than one application to consume documents produced by DirXML. Likewise, DirXML might tap into an existing message queue in order to consume its documents.

JMS Engine Processing

The DirXML engine processes the XML document by sequentially applying all configured rules based on the standard DirXML process flow. The driver can then manipulate the information using various rules, filters, and style sheets defined by the system administrator or developer. The driver then submits the data to eDirectory. When used with other DirXML drivers or driver instances, the data can be shared among many business applications and directories. Based on business rules, these other applications can add additional data that can, in turn, be submitted to other JMS-connected applications.

The DirXML Publisher Channel

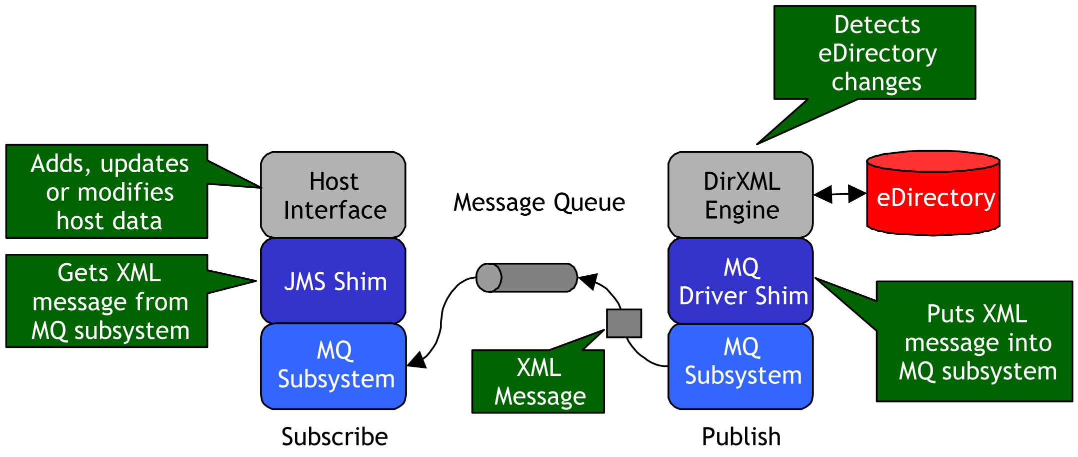
The Publisher   channel is used to integrate application data with eDirectory:


Adds or updates data in eDirectory
Configured to publish specific data
Host Interface
Message Queue
DirXML Engine
eDirectory
Puts message into MQ subsystem
JMS Shim
MQ Driver Shim
Gets and optionally transforms XML message
MQ Subsystem
XML Message
MQ Subsystem
Publish
Subscribe

The JMS shim publishes the application's output and converts it into XML document format and places the document into a specified JMS message queue or topic. The MQ Driver shim consumes the message from the specified queue or topic and submits XML-formatted changes to the DirXML engine for publication into eDirectory.

The DirXML Subscriber Channel

The Subscriber channel is used to integrate eDirectory data with host applications:


Detects eDirectory changes
Adds, updates or modifies host data
Host Interface
Message Queue
DirXML Engine
eDirectory
Gets XML message from MQ subsystem
JMS Shim
MQ Driver Shim
Puts XML message into MQ subsystem
MQ Subsystem
XML Message
MQ Subsystem
Subscribe
Publish

The Subscriber channel receives XML-formatted eDirectory events from the DirXML engine, converts these documents to an appropriate data format and publishes them to a JMS queue or topic where an application can consume them.

All eDirectory events that are filtered by the DirXML driver will be submitted to the JMS-interfaced application via the Subscriber channel. It is up to the application's JMS interface to interpret these messages correctly.

Publishing to eDirectory

When a JMS-connected application is determined to be an authoritative source of user profile data, that system can propagate all Add, Delete, and Modify object event data to eDirectory. The Publisher channel is used for propagation into eDirectory. For data to flow from the JMS-connected application to eDirectory, the driver utilizes the JMS interface to place XML documents into the appropriate queue for the application's interface.

JMS interfaces can be developed on a custom basis or purchased commercially as off-the-shelf connectors from 3rd parties. In addition, many third-party Extraction, Transformation and Loading (ETL) tools support the JMS interface standard and can readily be configured to communicate with many messaging middleware products. JMS ensures that an XML document is securely and reliably transported from the application or host system to eDirectory.

More information on using Java Message Service can be found at:

http://java.sun.com/products/jms/

Subscribing from eDirectory

The Subscriber channel of the driver is the component responsible for synchronizing data from eDirectory into a specific JMS message queue. This data can then be used to query, update and delete data managed by the host application.

Any combination of attributes, including those containing character string and binary data, can be subscribed from eDirectory via a JMS message queue. Additionally, JMS header information can be set by the driver and read by the subscribing application. The underlying JMS transport ensures that the XML payload is securely and reliably delivered from eDirectory to the host application.

Benefits

The driver uniquely enables the automation and maintenance of identity management processes with a very wide range of Enterprise applications:

- Message-based integration is inherently abstracted, which provides for greater data integrity and insulation against change. Greater abstraction also simplifies DirXML development, including collaboration and delegated work processes.

- Existing EAI infrastructure, which is increasingly found in public and private enterprises, can be fully leveraged for integration between DirXML and Applications and Data Warehouses.

- ERP applications often provide their own EAI and JMS messaging capabilities that can be fully leveraged at no additional purchase cost to the customer. For example integrating with Oracle Financials or Oracle Workflow can be performed using Oracle's Advanced Queuing (AQ) interface.
- Native connectivity to IBM S/390 Mainframe or AS/400 Midrange host platforms can be provided via IBM JMS middleware.

Features

The driver provides the following features:

- Support for JMS providers
- Support for Novell Remote Loader

Java-Enabled Enterprise Application Integration (EAI) Systems

- IBM* WebSphere* MQ
- Novell exteNd™ Enterprise
- OpenJMS
- Oracle* Advanced Queuing (AQ)
- SoftWired iBus Message Server*
- SpiritSoft SpiritWave TIBCO-JMS Bridge
- TIBCO JMS Server
- Support for additional JMS providers available upon request

Supported Platforms

- Linux*
- NetWare®
- Solaris*

- Windows* NT* 2000
- Windows 2003

JMS Driver Features

- Subscribe and Publish to Queues
- Subscribe and Publish to Topics
- Custom String Properties
- Message Priority
- TTL (Time To Live) JMS header.
- Query interface
- Publisher Tracking Interface
- Publisher Auditing Interface

Functional Channels operate independently (Examples: subscribe to both a queue and a topic simultaneously; publish to both a queue and topic simultaneously; publish to a queue and subscribe to a topic; publish to a topic and subscribe to a queue.)

For More Information about DirXML

For more information about DirXML, refer to the *DirXML Administration Guide* (http://www.novell.com/documentation/lg/dirxml11a/index.html) or the *Nsure Identity Manager Administration Guide* (http://www.novell.com/documentation/lg/dirxml20/index.html).

For more information about eDirectory, refer to the Novell® eDirectory documentation (http://www.novell.com/documentation/lg/edir871/index.html, or the documentation for a later version).

For more information about Java Message Service, refer to http://java.sun.com/products/jms/. For more information about specific message queue middleware, refer to your vendor's specific documentation.

JMS Configuration and Setup

You can integrate enterprise applications with the DirXML Driver for JMS to enhance and secure your organization's business processes. Before installing and configuring the driver, you must evaluate and design the processes that will underlie the integration. The design must define the driver's configuration, rules and style sheets to automate these processes as necessary. Additionally, queues or topics must be defined for your specific messaging middleware product.

Using iManager

Novell iManager is a tool for managing eDirectory. Additional administration and management capabilities are added to iManager through "plug-ins."

This new Web-based management tool was introduced with eDirectory 8.7. Because it is Web-based, you can do DirXML tasks from outside the firewall. Part of the iManager interface for DirXML is a helpful graphical representation of the rules and style sheets for each instance of the driver. Prior to installing and configuring the driver, you should install the DirXML plug-ins found on the DirXML CD.

If the version of DirXML you are using supports both administration tools, you can use both ConsoleOne and iManager to manage the same DirXML drivers. The tools are not mutually exclusive.

Product Components

The driver contains the following components:

- Driver configuration import files
- Driver shim

Driver Configuration

The driver configuration contains required driver parameters and policies. Driver Configuration files are provided with the driver for installation on a system that includes DirXML, JMS and the driver. Additional environments are also supported via manual configuration.

Driver Shim

The driver shim handles communication between the JMS message queue and the DirXML engine. It also allows for manipulation and querying of the JMS header for use of JMS-specific features by application developers and systems integrators.

Installing and Configuring the Driver

This section helps you do the following:

- Understand prerequisites for the driver
- Planning for installation
- Install driver components
- Configure driver parameters

Prerequisites

The DirXML® Driver for JMS requires the following:

- eDirectory™ 8.6.1. or higher
- Novell® DirXML® 1.1 or higher, or Identity Manager 2
- Novell iManager 1.5
- Java Virtual Machine (JVM*) 1.2 or higher
- Java Message Systems Libraries JMS.Jar

It is recommended that you create the Driver Set object before you install the driver. For more information about the Driver Set object, refer to Creating Driver Sets and Objects in the DirXML Administration Guide at: http://www.novell.com/documentation/lg/dirxml11a/index.html

Java Message Service-compatible messaging middleware, e.g. IBM JMS, TIBCO Rendezvous, Oracle AQ and Novell JBroker. JMS class libraries must be obtained from each vendor for its message queue implementation. (In some cases, there may be a licensing fee for the use of a vendor’s JMS classes. Be sure to verify licensing issues with your specific vendor as part of any project planning.)

iManager

Novell iManager 1.5 or later with the DirXML plug-ins installed

Supported Platforms

The driver runs on all DirXML-enabled platforms, including Windows* NT* 2000, NetWare®, Solaris*, and Linux*.

Planning for Installation

Before you install and use the driver, you must first plan for the installation.

JMS Checklist

- Establish or identify the queues that will be used either to hold event messages published to eDirectory and/or event messages subscribed from eDirectory

- Ensure that the appropriate vendor-specific JMS classes are installed on the server hosting the driver in accordance with the vendor's recommendation.

- Vendor-specific features: Ensure that any vendor-specific supporting components have been installed and configured properly on the JMS server. Example: when using topics with IBM WebsphereMQ, ensure that the message broker (strmqbrk) process been started.

Planning a Local or Remote Installation

This section explains the difference between a local and remote installation of the driver.

Local Installation

A local installation installs the driver on a server machine where you have JMS, DirXML and eDirectory installed.

Remote Installation

A remote installation installs the driver on a different computer than the one where DirXML and eDirectory are installed. When using NetWare, you may want to use this type of installation when installing the JMS driver with if your JMS version is not supported to run under NetWare.

Installing Components

The driver installation program installs the following components on the server:

Component	Description
Driver Shim	A java driver shim that communicates between the JMS message bus and the DirXML engine.
Driver Configuration DirXML Driver	All eDirectory objects, including the appropriate rules and style sheets for adding, modifying, and deleting or disabling objects. Controls the information being sent from JMS to eDirectory and from eDirectory to JMS.

Although the installation program installs the components, setup is not complete until you properly configure the Driver object and the JMS system.

Installing or Upgrading The Driver

This section will help you use the Application Driver Creation Wizard to install and configure the driver.

Installing the Driver

1. Download JMS_MQ_Install.exe or JMS_MQ_Install.bin and run it.
2. Run INSTALL.EXE found in the product distribution.
3. Accept the license agreement > click Next.
4. Select the components you want to install and the directory paths that each component will be installed to > click Next.

Install the Driver Configuration files on the system.

Component	Filename
Driver Shim	Jms.jar (supporting fils) Jmsdriver.jar
Drive Configuration File	Jmsdriver20.xml

Click Finish.

The Install packaged will copy the files Jms.jar, Jmsdriver.jar and jmsdriver20.xml to the default \novell\nds\lib directory.

To complete the installation manually copy the jmsdriver20.xml file to the appropriate directory. If you are using ConsoleOne®, copy the file to the **novell\consoleone\1.2\snapins\dirxml.** If you are using Novell iManager, copy the file to **tomcat/4/webapps/nps/dirxml.drivers/**

Upgrading from 1.0 to 2.0

After you download the CD image, perform the following steps to upgrade a previous version of the driver:

1. Stop the drivers being upgraded. Select Manual for the driver’s startup option.
2. Stop eDirectory.
3. Copy the JMSDriver.jar and JMS.Jar into the appropriate directory for your platform Use the following table to determine the appropriate directory

Platform Directory Path

Platform	Directory Path
NetWare®	SYS:\SYSTEM\LIB
Solaris or Linux	/usr/lib/dirxml/classes
Windows NT/2000	NOVELL\NDS\LIB

1. Restart eDirectory.
2. (Optional) Install the driver configuration.
3. Export a copy of the Current Driver Configuration
4. In the Drive Module Replace the com.novell.nds.dirxml.driver.mq.MqDirXMLDriverShim with com.novell.nds.dirxml.driver.JMS.JMSDirXMLDriverShim
5. Make a Copy of the Current Driver Parameters
6. Replace the Driver Parameters with the New Driver Parameters Shown Below:

```
<?xml version="1.0" encoding="UTF-8"?>
<driver-config name="JMS Driver">
        <driver-options>
                <qmgr display-name="Queue Manager">QM_identric</qmgr>
                <msgbrk display-name="Message Broker"/>
                <mqchannel display-name="WebSphere MQ Channel"/>
                <sp display-name="Security Principal"/>
                <purl display-name="Provider Url"/>
                <sc display-name="Security Credentials"/>
                <ss display-name="Security Protocol"/>
                <vb display-name="Verbose">TRUE</vb>
                <port display-name="OraclePort"/>
                <osid display-name="OracleSID"/>
                <ouser display-name="Oracle-Schema"/>
                <otable display-name="OracleQueueTable"/>
                <clientid display-name="JMS Client ID"/>
                <vendor display-name="Vendor">MQSeries</vendor>
        </driver-options>
        <subscriber-options>
                <qsend display-name="Queue Sender">subq</qsend>
                <tpub display-name="Topic Sender"/>
                <nonper display-name="Message persistence"/>
                <bytemessaging display-name="Byte Messaging"/>
                <streammessaging display-name="Stream Messaging"/>
                <jmsclient display-name="Client  Mode"/>
                <subqueryq display-name="Subscriber Query Queue"/>
                <subqueryreplyq display-name="Subscriber Query Reply Queue"/>
                <subquerytimeout display-name=" Subscriber Query Reply Timeout"/>
                <ttl display-name="JMS Time To Live"/>
                <pri display-name="JMS Priority"/>
                <marker display-name="Remove Scripting Markers">True</marker>
        </subscriber-options>
        <publisher-options>
                <qrec display-name="Publisher Queue Receiver"/>
                <queryq display-name="Publisher Query Queue"/>
                <queryreplyq display-name="Publisher Query Reply Queue"/>
                <querytimeout display-name="Publisher Query Reply Timeout"/>
                <trec display-name="Topic Receiver"/>
                <headers display-name="Retrieve Headers"/>
                <jmsclient display-name="Client  Mode"/>
                <auditq display-name="Error Auditing Queue"/>
                <trackingq display-name="Transaction Tracking Queue"/>
                <correlationid display-name="Use Correlation ID'S"/>
        </publisher-options>
</driver-config>
```

7. Set the Client Mode to MQ if you need to have native mode JMS Messages.
8. Set the driver's startup options to their previous values.
9. Restart the drivers.

Activating DirXML Products

DirXML, Identity Manager, and DirXML drivers must be activated within 90 days of installation, otherwise they will shut down. At any time during the 90 days, or afterward, you can choose to activate the products to a fully licensed state.

NOTE: *Activating a driver does not change your current configuration or install a newer version of the driver shim. It simply changes the driver to an activated state.*

The following examples describe various activation scenarios you might encounter:

- You purchase a DirXML bundled activation. This includes the activation of multiple drivers and the DirXML engine.
- You purchase individual driver activation for DirXMl. This includes the activation of a single driver and the DirXML engine.
- You purchase driver group activation for Identity Manager.
- You purchase customized or third party driver activation. This includes the activation of a customized or third-party driver and the DirXML engine.

For information about activating with DirXML, refer to Activating DirXML Products
(http://www.novell.com/documentation/lg/dirxml11a/dirxml/data/agoppxb.html) and Viewing Product Activations for DirXML and DirXML Drivers
(http://www.novell.com/documentation/lg/dirxml11a/dirxml/data/agoppxb.html#agfhtax).

For information about activating with Identity Manager, refer to Activating Novell Identity Manager Products
(http://www.novell.com/documentation/lg/dirxml20/admin/data/afbx4oc.html).

For additional information about Activation, refer to Activation Basics
(http://www.novell.com/partners/partnerplace/epd/product_activation_basics

.html) and Activation Troubleshooting (http://www.novell.com/partners/partnerplace/epd/troubleshooting_activation.html).

To purchase DirXML licenses, see the information on How to Buy (http://www.novell.com/products/edirectory/dirxml/howtobuy.html).

Post Installation Tasks

Now that you have installed the driver, you will need to do the following:

Import the Driver Configuration file.

Configure the Remote Loader on the remote system.

Importing the Driver Configuration

The Application Driver Creation Wizard will help you import a Driver Configuration file. This file will create and configure objects needed in eDirectory to make the driver work properly.

Right-click Network Neighborhood > click NetWare Connections.

Ensure the following:

- You are authenticated to the eDirectory server you are installing to.

- The server where the driver will be running is set as the primary server.

In IManager, click Wizards > Create a New Application Driver.

Select the driver set where you want the driver installed > click Next.

Enter the following driver set properties as prompted:

- Name
- Context
- Whether to create a new partition on the driver set

Select Jmsdriver.xml driver configuration file > click Next.

Follow on screen prompts to enter information about eDirectory structure and driver connectivity.

Click Next and enter the following information as prompted:

- The name you want to use for the driver

- The driver password

The shim authentication password is used for logging into JMS. The driver password is used to authenticate to the DirXML server.

When the driver import is finished, click Yes to define security equivalences on the driver.

- Click Add > select an object with Admin rights (or any other rights you want the driver to have)

- Click Apply > Close.

1. Exclude Administrative Roles from replication > click Apply > click Close.

2. Click Finish.

Configuring the Remote Loader on the Remote System

The optional Remote Loader can be configured to use a SSL connection for secure data transfer between eDirectory and the JMS system. This section explains what needs to be completed to establish an SSL connection.

Generate a security certificate on the eDirectory container.

Create an organizational trusted root certificate.

Validate the certificate file.

Run Remote Loader wizard and configure options.

Configure Driver Object properties in iManager.

Generating a Security Certificate on the eDirectory Container

In iManager, select the Container in which the eDirectory server resides.

Right-click the container > select New Object > create the NDSPKI: Key Material object.

Enter a name for the certificate > select the creation method > click Next.

IMPORTANT: Write down the name of the certificate. You will need this information when configuring the driver.

Choose the Standard Creation Method > click Next to view the Confirmation Screen.

Click Finish to generate a certificate for the eDirectory container.

Create an Organizational Trusted Root Certificate

In IManager, browse to the Security container and locate the Organizational CA for the tree.

Right-click the certificate object > click Properties.

Click the Certificate tab and go to the Self-Signed Certificate option.

Select Export > click Next.

Select the Base64 option > click Next.

Click Finish.

Click Validate to verify the Certificate's validity.

Copy the certificate file to the Remote Loader directory on the Remote Loader host system.

Using the Wizard to Configure Remote Loader Properties on the Remote Loader Host Server

From a DOS command prompt on the Remote Loader host system, change directory to the Remote Loader directory and enter: dirxml_remote

Click Next to begin the wizard.

Following the wizard prompts, you should configure the following items:

- Command Port. Used to differentiate between instances of Remote Loader.

- Increments. The first instance of Remote Loader defaults to port 8000 with each additional instance incrementing in number by one.

- Configuration File. The configuration file is a filename that is automatically generated. Verify the filename > click Next.

DirXML Driver. Select the Java option. Enter the case-sensitive java class name: com.novell.nds.dirxml.driver.JMS.JMSDirXMLDriverShim.

- Connection to DirXML. Enter the port number. The default is 8090, but you can enter a different, unique port number. Select the IP address to which you want to apply changes. You should select the All IP addresses option.

- Certificate. Select Use SSL. Browse to the certificate generated in the previous section. The browser automatically looks for Base64 files > click Next.

- Tracing. Choose a level of tracing. In addition to trace files, you can log trace events to a log file. Configure this log file if you want. Trace level 0 provides no trace application window.

- Install as a service. When you install Remote Loader as a service, it will automatically launch when the machine boots.

- Passwords. Enter a password for the remote loader and driver object.

IMPORTANT: Write down these passwords. You will need them for the next step in driver configuration.

- Summary. Click Next to continue with the DirXML Remote Loader configuration.

Click Finish to complete the configuration. When prompted to start the DirXML Remote Loader, click Yes.

Starting the Remote Loader if Installed as a Service

If you configured Remote Loader to run as a service, you can start it by completing the following steps:

Open the Windows Service Control Manager.

Select the DirXML Loader (com.novell.nds.dirxmldriver) > click the Startup Option > click Start.

Starting the Remote Loader Manually

From a command prompt (wherever Remote Loader is installed), enter:

```
-dir *.txt
```

This will display the name of the configuration file (for example, config8001.txt).

Enter the following:

```
dirxml_remote config <configfilename.txt>
```

For example:

```
dirxml_remote -config config8001.txt
```

You can open the Windows Task Manager to verify that Remote Loader is running.

Note: If the Java Message Service factory classes have been installed into eDirectory, the driver retrieves these classes from the directory prior to initialization. If the remote driver exists in a DMZ or is separated by a firewall, the default TCP ports 8090 for the remote loader, 389 for LDAP and optionally 636 for LDAP/S must be open to the server that hosts the driver.

Configuring Driver Object Properties for Remote Loader

In iManager, select the JMS Driver object > click Properties.

Click the Authentication tab.

Enter the following Remote Loader connection parameters with a space between each parameter:

Hostname. Specifies the address or name of the machine on which the Remote Loader will run. For example, hostname=192.168.0.1

Port. Specifies the port on which the Remote Loader will accept connections from the remote interface shim. For example, port=8090

KMO. Specifies the key name of the Key Material Object containing the keys and certificate used for SSL. For example: kmo=remotedrivercert.

NOTE: If SSL is enabled, you need to enter the hostname, port, and KMO information.

If SSL is not enabled, you only need to enter the hostname and port information.

Enter passwords for the application and the Remote Loader.

Click the Driver Parameters tab > scroll to the Publisher Settings.

Verify that the Publisher queue is configured. This queue contains XML documents that will be processed against eDirectory. Refer to Configuring Driver Object Properties for more information.

Configuring The JMS Driver Parameters

This section helps you to configure the Driver object properties. In addition to the following driver-specific configuration parameters, you should configure basic DirXML Driver object fields. For more information regarding these basic configuration fields, see the DirXML Administration Guide.

Configuring Driver Parameters

In iManager, the Driver object > click Properties > Properties.
Configure the following:

Tabs/ Parameters	Description
Driver Module	
JMS	This is the type and name of the DirXML driver: com.novell.nds.dirxml.driver.JMS.JMSDirXMLDriverShim
Authentication	
Authentication ID	This is the User name to Authenticate against the JMS provider installation
Authentication Context	Optional directory context of the authentication object.
Application Pass-word	Password used in conjunction with the Authentication ID.
Driver Settings	
Queue Manager	This element specifies the name of a queue manager.
Message Broker	This element specifies the name of message broker. For Non JMS Implementations this field must be filled in with the element Message Broker if you wish to send or receive using topics
Security Principal	This element specifies the DN of a user that has the rights to access the NDS server to retrieve either the JMSTopicConnec-tionFactory or JMSQueueConnectionFactory object.
Security Credentials	This element specifies the password of the user DN contained in the sp element.
Security Protocol	This element enables an ssl connection to be created while connecting to the NDS server. The only valid value is 'ssl', oth-erwise leave empty.
Verbose	This element enables verbose logging to the DirXML trace to enhance debugging. The only valid value is 'true', otherwise leave empty.

OraclePort	This Element is used only for connecting to Oracle Advanced Queueing. The Element indicate the oracle port listening for connections.
Oracle SID	This Element is used only when connecting to Oracle Advanced Queuing. The Element Represents the Oracle SID Identifier.
Oracle-Schema	This Element is used only when connecting to Oracle Advanced Queuing. This Element is used to specify the Schema Tag the queues reside under.
OracleQueueTable	This Element is used only when connecting to Oracle Advanced Queuing. This Element represents the actual queue table name that will hold the JMS Messages
JMS Client ID	This Element Specifies the JMS Client identifier used to connect to a queue or topic. If left blank the Shim will generate a random JMS Client ID
Vendor	This Element specifies the JMS Vendor Implementation Valid Values are (MQSeries, Oracle, Tibco, TIBCOJMS, Jbroker, OpenJMS and IBUS
Subscriber Settings	
Queue Sender	This element specifies the name of the queue to send DirXML Messages
Query Queue	This element specifies the queue that will be used to query the remote application via JMS
Query Reply Queue	This element specifies the queue that will be used receive the replay message from the remote application. The query the remote application via JMS
Query Reply Queue Timeout Period	This element specifies the number of seconds the Subscriber channel will wait for the query reply message from the JMS application. This value has no effect when enabled with pub/sub messaging. The only valid value is an integer value for seconds, otherwise leave empty for the default value of 10 seconds to be used.
Topic Sender	This element specifies the topic to publish NDS event information to.
Client Mode	This element specifies the mode or style to use in placing Messages on a Queue or topic. The valid styles are WebsphereMQ native message or JMS Messages The only valid values are blank or MQ.
Byte Messaging	Bytes Messaging is used to send a message containing a stream of uninterrupted bytes. The receiver of the message is responsible for the interpretation of the bytes. This should only be used if DirXML needs to exchange non-text messages with your application.

	Input True for sending a byte stream, or leave blank for the default (text) option.
JMS Time To Live	This element specifies the maximum lifetime for a message. The input is expected to be in milliseconds. If the element is blank then the message has an unlimited lifetime.
JMS Priority	This element specifies the Priority of the message. Valid values are from 0 through 9. If the element is left blank the message priority default to 4.
Message Persistence	This element specifies whether messages will be non-persistent. Default is persistent. The only valid value is 'true', otherwise leave empty. (Does this mean that setting to 'true' makes the message non-persistent? If so, we should rename the item to Message Non-Persistence…!)
Publisher Settings	
Topic Receiver	This element specifies a JMS queue to receive XML formatted
Queue Receiver	This element specifies a JMS queue to receive XML formatted DirXML commands from.
Query Queue	This element specifies the queue that will be used to query the remote application via JMS
Query Reply Queue	This element specifies the queue that will be used receive the replay message from the remote application. The query the remote application via JMS
Query Reply Queue Timeout Period	This element specifies the number of seconds the Subscriber channel will wait for the query reply message from the JMS application. This value has no effect when enabled with pub/sub messaging. The only valid value is an integer value for seconds, otherwise leave empty for the default value of 10 seconds to be used.
Audit Queue	This element specifies a JMS queue to transaction returned an error.
Tracking Queue	This element specifies a JMS queue to place a copy of the message that was successfully processed by the DirXML engine.
Use Correlation ID'S	This element specifies whether to append a correlation identifier to both the tracking and Audit queue Messages.

Troubleshooting the Driver

This section contains potential problems and error codes you may encounter while configuring or using the driver.

Using the DSTrace Utility

You can troubleshoot the driver using the DSTrace utility. You will want to configure the utility's options by selecting Edit > Properties > DirXML Drivers.

For each event or operation received, the driver returns an XML document containing a status report. If the operation or event is not successful, the status report also contains a reason, a text message describing the error condition. If the result is fatal, the driver shuts down.

Common Errors

Driver Load Errors	**Solution**
java.lang.ClassNotFoundException: com.novell.nds.dirxml.driver.JMS.JMSDirXMLDriverShim	This is a fatal error that occurs when JMSdirXMLDriverShim.jar or JMS.Jar is not installed properly. You should ensure that the file is in the proper location for either a local or remote loader configuration.
java.lang.ClassNotFoundException: com.novell.nds.dirxml.driver.JMS.JMSDirXMLDriverShim	This is a fatal error that occurs when the class name for the JMSDirXMLDriverShim.jar is incorrect. You should ensure that the Java class name is set on the Driver Module tab in a local installation and that the -class parameter is set in a remote loader configuration. The proper class name is: com.novell.nds.dirxml.driver.JMS.JMSDirXMLDriverShim
-295	Almost all-295 errors indicate a missing supporting java class library
Driver Initialization Errors	**Solution**
java.lang.ArrayIndexOutOfBoundsException # > #	There is a problem w/the JDK version when trying to retrieve JMS QueueConnectionFactory object via JNDI. This does not apply to a local JMS connection. Replace the <Novell_Install_Path>\NDS\jre\lib\rt.jar with a more JDK version 1.3.1 and above rt.jar
'no mqjbnd02 in java.library.path'	Ensure that the <MQSeries_Install_Path>\java\lib is in your System path.
MQJMS2008: failed to open MQ queue. MQJMS2005: failed to create MQQueueManager for <>	Ensure the specified queue in the 'Driver Parameters Xml' exists on the WebsphereMQ server. Ensure the WebsphereMQ server is running.
MQJMS2006: MQ problem: com.ibm.mq.MQException: Completion Code 2, Reason 2085	Regarding MQ JMS Pub/Sub calls, ensure to create the mq jms system queues. Example: C:\<MQSeries_Install_Path>\java\bin\runmqsc <queue_manager> < MQJMS_PSQ.mqsc
-295 errors	Almost all-295 errors indicate a missing supporting java class library
-295 javax/transaction/xa/XAException	The JVM 1.4 requires the use of the JTA classes available from java.sun.com
mqjms1046: The character set 437 is not supported.	Set the sending client ccsid to 819 (the default).

Developing Solutions using DirXML Driver for JMS

JMS-enabled applications that need to exchange data with the driver must use messages created in XML format. The driver will pass XML messages to the DirXML engine and return the results in XML.

The desired XML message format should comply with Novell's nds.dtd document type definition. If your JMS application is unable to work with the XML format specified within the nds.dtd, it can use any style of XML, however, DirXML transformation style sheets must be created to allow the driver to do the conversion of data from one format to another.

Note for JMS Users: the driver for JMS can work with either MQ Series style messages or regular JMS style messages. However, each instance of the driver for JMS can only work with one type of message.

Using JMS Features

In addition to the configurable parameters, the driver supports the use of many additional Java Message Service features that enhance the developer's ability to create solutions.

Message Properties.

A JMS Message contains a built-in facility for supporting application defined property values. This provides a mechanism for adding application specific header fields to a message. Properties allow an application, via message selectors, to have a JMS provider select/filter messages on its behalf using application-specific criteria:

Custom Message properties can be set through a DirXML Style sheet. This method is needed to pass name value pairs to systems that are JMS header dependant.

An Example of setting a custom header is shown below. We will show you how to set a customer header of Last Name to Smith

{LastName|Smith}

The JMS Shim looks for name value pairs contained by brackets { } and separated by a '|'/

{FieldName|FieldValue}

Setting Binary Values in JMS Message.

The JMS Driver 2.0 support setting binary content in Subscriber channel Messages. Many legacy application use nonprintable ASCII characters as record delimiters. The current XSLT specification does not contain a mechanism for handling non-printable characters natively.

Custom Binary characters can be set by using specific markers the JMS Driver know how to interpret. This method is needed to pass binary value pairs to systems that depend on nonprintable characters for their record delimiters.

An Example of setting a Binary Value is shown below. We will show you how to set a Binary Value or 255 or FF.

{#CC|255}

The JMS Shim looks for name value pairs contained by brackets { } and separated by a '|'/

{FieldName|FieldValue}

In this particular case the JMS Shim know that the specific FieldName of #cc indicates it will insert a character of (FieldValue) in the message. The FieldValue Is expressed in integer format so if you wanted to insert an character of FF equivalent you would express this as {#CC|255}

Message Priority.

JMS defines a 10 level priority value with 0 as the lowest and 9 as the highest. JMS clients or DirXML style sheet logic should consider 0-4 as lower –to-normal priority and 5-9 as normal-to-higher priority. Priority is set to 4, by default.

TTL (Time To Live).

JMS defines the length of time in milliseconds that a message should be retained by the message system. Time to live is set to zero by default.

JMSMessageID.

A JMS Message ID property contains an optional value that uniquely identifies each message sent by a provider. This can be used to historically identify each message or in concert with Correlation ID to establish a relationship among requests and replies in an asynchronous messaging environment.

Correlation ID.

The JMS CorrelationID property is used for linking one message with another. It typically links the Message ID of the reply message with its Correlation ID of the requesting message.

Associations

Associations are made between messages and eDirectory objects through a style sheet. A unique ID can be created for records relating to each system connected via JMS.

The association attribute received from a queue is unique to the interfaced application, based on each driver instance that you install and enable.

If other drivers are installed, they should use an association specific to that application.

The association attribute is multi-valued. Therefore, if DirXML is being used to connect multiple applications, all of their associations can be stored on this attribute.

The unique ID association links objects in the JMS-connected application to their objects in eDirectory. When an ADD event occurs, the association is made and referenced in the style sheets. This association allows the driver to perform subsequent tasks on the appropriate object.

If a User object is added in eDirectory, the unique user's ID might be populated so that the ID can be linked to the individual's record in the JMS-connected application. The matching rule is executed and, if the ID is found within the application, the association is then created. The Matching rule matches the eDirectory user ID and the appropriate unique ID on the host system.

Setting the Global Association Attribute Value

A global association attribute value can be set by modifying a value in the command transformation stylesheet.

The association value can be statically set by modifying the association state element, or dynamically computing this value based on additional logic that you can add to the stylesheet to meet an application's requirements.

Overriding the Global Association Attribute Value

The global association value that you establish in the stylesheet can be dynamically overridden within a Publisher channel message. This is a powerful feature that is made possible by the open-ended nature of the driver's design.

To override the global value, you must include an association attribute value inside an XDS message submitted to the Publisher channel as follows:

<association> value <association/>

The role of the Subscriber Channel in Setting Associations

In the JMS driver, the Subscriber channel plays an important role is setting the association values. At first, this may seem counter-intuitive, but it takes advantageof DirXML engine features and is very efficient.

Due to nature of marrying DirXML to the flexible and asynchronous JMS interface, the driver automatically creates a dummy association for each add event. This association value is then updated according to application needs, either by setting a global, static value in the spreadsheet or dynamically within a Publisher channel message (as described above).

When setting the association attribute value using the stylesheet, a filter for the Subscriber channel must be enabled. This allows the relevant XSL script to fire in order to correctly update the association values. When setting the association attribute value using an XDS message for the Publisher

Association Style Sheets

Overview

There are 3 primary ways to set an association value with the JMS Driver.

- Subscriber Channel [Outgoing Document is in XDS format]
- Subscriber Channel [Outgoing Document is in NON XDS format]
- Publisher Channel

Association StyleSheet (XDS Format)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:template match="add">
		<xsl:copy>
			<xsl:apply-templates select="@*"/>
			<association state="associated">MQAssociation</association>
			<xsl:apply-templates select="node()"/>
		</xsl:copy>
	</xsl:template>
	<xsl:template match="node()|@*">
		<xsl:copy>
			<xsl:apply-templates select="node()|@*"/>
		</xsl:copy>
	</xsl:template>
</xsl:stylesheet>
```

Association Write Back (Non XDS Format)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:cmd="http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.XdsCommandProcessor"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:strip-space elements="*"/>
	<xsl:preserve-space elements="value component"/>
	<xsl:output indent="yes" method="xml"/>
	<!-- ****************** -->
	<!-- DirXML passed parameter for cmd processing -->
	<!-- ****************** -->
	<xsl:param name="srcCommandProcessor"/>
	<!-- ****************** -->
	<!-- Global Variables. -->
	<!-- ****************** -->
	<xsl:variable name="debug">false</xsl:variable>
	<xsl:variable name="message">true</xsl:variable>
	<!-- ****************** -->
	<!-- ****************** -->
	<xsl:template match="add">
		<!-- convert modify or sync with an association to an instance so that -->
		<!-- output transform can create a complete output record -->
		<xsl:variable name="associationValue" select="string(association/text())"/>
```

```
                <xsl:choose>
                        <xsl:when test="association[@state = 'disabled']">
                                <!-- ignore if the association is disabled -->
                        </xsl:when>
                        <xsl:otherwise>
                <!-- if a modify on an associated object the association replace it with the instance -->
                                <!-- returned by querying the object -->
                                <xsl:variable name="assoc-cmd">
                                <add-association dest-dn="{@src-dn}" dest-entry-id="{@src-entry-id}">
                                                <xsl:value-of select="JMSAssociation"/>
                                        </add-association>
                                </xsl:variable>
                                <!-- query NDS variable can be called anything -->
<xsl:variable name="do-cmd" select="cmd:execute($srcCommandProcessor, $assoc-cmd)"
xmlns:cmd="http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.XdsCommandProcessor"/
>
                        </xsl:otherwise>
                </xsl:choose>
        </xsl:template>
</xsl:stylesheet>
```

Association (Publisher Channel)

To set the association via the Publisher channel you simply need to add an association tag into the XDS document as shown below.

```
<association state="associated">MQAssociation</association>
```

Channel Write Back

Overview

The JMS driver has support for the Channel Write Back on both the publisher and subscriber channels.

When using the channel write back to send messages to the JMS provider you must tell the command processor the name of the queue. You must specify the name of the queue by placing the following string "{cmdQueue|Queue Name}" at the beginning of the variable. The command processor will remove the special command tags when processing the message.

The command processor invocation automatically encapsulates the message with the "<nds><input> </input></nds>" tags. You can instruct the command processor to remove the tage by including the following command in the variable definition "{RemoveInputTags|True}". A more in depth discussion about how to use the Channel write back functionality is included below.

XSLT Channel Write-Back In-Depth

Now that you've seen a simple example of XSLT Channel Write-Back, let's dig in deeper and understand how it all works. There are four items that Channel Write-Back uses in any stylesheet:

- Name Space Declarations
- Parameter Declarations
- XDS Fragment Document
- Channel Write-Back Execution

Name Space Declaration

The first item you need to add is a name space declaration that instructs Novell's XSLT processor to bind the prefix "cmd" to a Java class that implements the Channel Write-Back functionality.

xmlns:cmd="http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.
XdsCommandProcessor"

Typically, the *<xsl:stylesheet>* element's start tag will look something like this:
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
To enable Channel Write-Back in a stylesheet, the *<xsl:stylesheet>* element's start tag should look like this:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:cmd="http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.
XdsCommandProcessor" >
```

Note that in terms of DirXML, the <*xsl:stylesheet*> and <*xsl:transform*> are identical and can be used interchangeably.

Parameter Declaration

In addition to adding the name space declaration, two stylesheet parameters must be declared:

- SrcCommandProcessor
- destCommandyProcessor

These two parameters are passed by the DirXML engine to the stylesheet. They are used to allow the stylesheet to specify which datastore should be written back to (modified). The terms "source" and "destination" are used in context of the origin of the event. In other words, if you need to query the datastore that generated the event, you send the XDS Fragment query to the srcCommand- Processor. If you need to modify the datastore that will be receiving the event, you send the XDS Fragment to the destCommandProcessor.

Here's a hint that might help you. Since the Publisher Channel is always used for events coming into eDirectory, the destination datastore will always be eDirectory. Therefore, the destCommandProcessor is used to modify eDirectory on the Publisher Channel. Conversely, since the Subscriber Channel is always used for events coming from eDirectory, the source datastore will always be eDirectory so the srcCommand-Processor is used to modify eDirectory.

These two parameters should be declared towards the top of your stylesheet, after the <*xsl:stylesheet*> element. While it is only necessary to declare the parameters that are actually used, it's good practice to declare them both:

```
<xsl:param name="srcCommandProcessor"/>
<xsl:param name="destCommandProcessor"/>
```

The XDS Fragment Document

Now that you've properly prepared your stylesheet for Channel Write-Back, you can now delve into the actual Channel Write-Back itself. The structure of the XDS Fragment document is the same as any XDS document. This means it must comply with the NDS.DTD. The full details of the XDS Document structure can be found in the NDS.DTD file located at http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/ndsdtd/query.html.

Once the XDS Fragment Document is constructed, it is typically stored in an <*xsl:variable*> so that it can be passed to the Command Processor for execution.
Sample XDS Fragment Documents
Below are some sample XDS Fragment Documents to better illustrate how to properly construct such a document.

Sample 1. This is a document to modify a user's Description attribute.

```
<modify class-name="User" dest-dn="\AMCE\Users\Sam">
<association>JMS Assocation<association>
<modify-attr attr-name="Description">
<remove-all-values/>
<add-value>
<value>This is a new Description</value>
</add-value>
</modify-attr>
</modify>
```

Sample 2. This is a document will send a message to a queue names TestQueue remove an object's association. Please note the content of the variable do not have to be in xml format. They contents can be in any format needed or desired.

```
<xsl:variable name="TextMessage">
 {cmdQueue| TestQueue }{RemoveInputTags|True}
   <TextMessage>
               <Test text message/>
  </ TextMessage >
</xsl:variable>
```

Sample 3. This is a document to add an Alias to an object.

```
<add class-name="Alias" dest-dn="\ACME\Aliases\Sam">
<add-attr attr-name="Aliased Object Name">
<value type="dn">
<xsl:value-of select="\ACME\Users\Sam"/>
</value>
</add-attr>
</add>
```

Sample 4. Examples of a Non XML formatted message

```
<xsl:variable name="TextMessage">
 {cmdQueue| TestQueue }{RemoveInputTags|True}
   This is a test Text Message
</xsl:variable>
```

The Channel Write-Back Execution

Once you've properly constructed your XDS Fragment Document and stored it in an <*xsl:variable*> called *$cmd-Update*, you need to submit it to the Command Processor for execution. There are a few components to the execution:
The name of the variable that will contain the results (unlike with queries, the result variable is never used after execution; it is used to execute the command)
The *cmd:* prefix to indicate that this command will be handled by the Command Processor as declared in the Name Space Declaration

A reference to either the *srcCommandProcessor* or the *destCommand- Processor* to instruct the DirXML engine as to which datastore the XDS Fragment should be applied to A reference to the variable in which the XDS Fragment Document is stored
When you put these elements together, you get the following command:
<xsl:variable name="result" select="cmd:execute($srcCommandProcessor, $cmd-Update)"/>
$result is the <*xsl:variable*> you will store the Channel Write-Back results in. This variable has no function once the command is executed.

Query interface

Overview

The JMS driver has two query interfaces; one on the Subscriber channel and one on the Publisher channel

Each query interface has three components: Query Queue, Query Reply Queue and a Query Timeout value. The query interface uses a synchronous transaction mode.

The query reply queue interface operates in blocking (synchronous) mode. When the query interface in invoked, it places a query message on the query queue and then waits for a reply to this message on the designated query reply queue until the value of the timeout has been exceeded.

- Query Queue
- Query Reply Queue
- Query Timeout Value

Query Queue

The Query Queue defines the queue that query documents will be placed on. Typi-cally these documents are in XML or XDS format but can be in any format. The re-mote JMS client is then responsible for interpreting this query document then placing the response to the query on the Query Reply Queue.

Query Reply Queue

The Query Reply Queue interface defines the actual queue used for the query reply documents that are sent by the remote JMS client. This interface is used to which document to respond to by listening for a message with the JMS Correlationid set to the message ID of the original query document. The remote JMS client is responsible for receiving the original query message and setting the JMS Correlation ID equal to the JMS Message ID

Query Timeout Value

The Query Timeout specifies the maximum length of time the query reply queue interface will wait to receive a reply from the remote JMS client responsible for responding to query messages. No other DirXML processing will occur during the specified period (a.k.a blocking). If this field is left blank it defaults to 10 milliseconds.

Sample Query Style Sheet

```
<xsl:template name="query-object-name">
<xsl:param name="object-name"/>
 <!-- build an xds query as a result tree fragment -->
 <xsl:variable name="query">
	<nds dtdversion="1.0" ndsversion="8.5">
	<input>
		<query>
	<search-class class-name="{ancestor-or-self::add/@class-name}"/>
	<!-- NOTE: depends on CN being the naming attribute -->
	<search-attr attr-name="CN">
		<value>
			<xsl:value-of select="$object-name"/>
		</value>
	</search-attr>
		<!-- put an empty read attribute so only object is returned. -->
	<read-attr/>
	</query>
	</input>
	</nds>
 </xsl:variable>

<!-- query eDirectory -->

 <xsl:variable name="result" select="query:query($destQueryProcessor,$query)"/>
 <!-- return an empty or non-empty result tree fragment depending on result of query
-->
	<xsl:value-of select="$result//instance/@class-name"/>
 </xsl:template>
```

Sample Query Reply Documents

Included below are sample document representing the default format for the returned query documents. However, returned query document can be in any desired format.

One Instance Found.

If a single object is found based on your search criteria, the following document will be returned:

```
<instance class-name="User" event-id="0" src-dn="\TEST_TREE\TEST\Users\JDoe">
<association state="associated">JDoe@test.com</association>
    <attr attr-name="Surname">
         <value timestamp="1016241258#16" type="string">Doe</value>
     </attr>
```

```
</instance>
```

Notice that since only a single object matched your search criteria, there is a single <*instance*> element in the document above.

Multiple Instances Found.

The default format for the returned query document when the search returns multiple results. However, this is only the default format and you are free to return the document in any form you wish.

```
<instance class-name="User" event-id="0" src-dn="\TEST_TREE\TEST\Users\JDoe">
<association state="associated">JDoe@test.com</association>
    <attr attr-name="Surname">
        <value type="string">Doe</value>
    </attr>
</instance>
<instance class-name="User" event-id="0" src-dn="\TEST_TREE\TEST\Users\JDoe2">
<association state="associated">Jdoe2@test.com</association>
    <attr attr-name="Surname">
        <value type="string">Doe</value>
    </attr>
</instance>
```

Audit Features

The JMS driver provides a mechanism that helps to ensure best practices for handling transactions submitted to the Publisher channel.   The features are:

- Audit Queue Interface
- Tracking Queue

Audit Queue

When documents are submitted to the Publisher channel but then fails to be processed for any reason, the Audit Queue Interface places a copy of the original message and a copy of the returned error message on the indicated Queue.

Tracking Queue

The Tracking Queue interface captures all messages that processed successfully and places a copy of the message on the tracking queue.

Automatic Fail Over and Load Balancing

Multiple JMS drivers may be configured to retrieve messages from a single queue. Under this configuration, the queue feeds messages to each attached driver in a "round-robin manner.

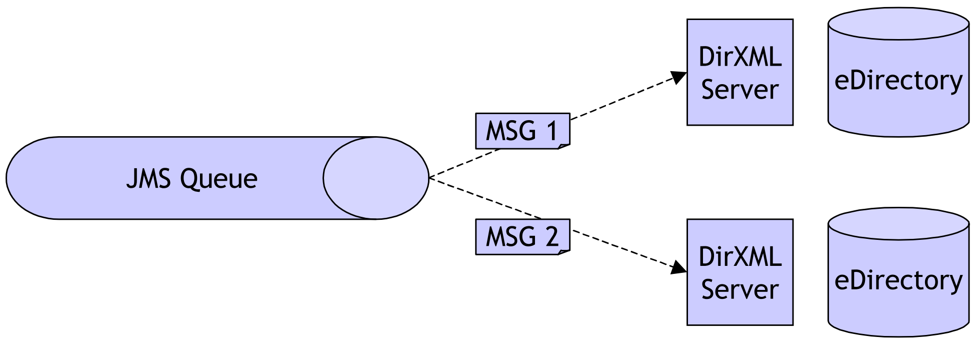
Automatic Fail-Over

Fully automatic fail-over can be supported by configuring two JMS drivers that fetch messages from a single queue (or queue cluster). In the event that one of the drivers fails for any reason, the second will automatically continue processing. No messages can be lost during this process.

Load Balancing

When configured on multiple DirXML servers, the JMS driver supports true load balancing that can greatly scale the performance of DirXML. Where XML transaction volumes exceed the ability of the engine to keep pace, simply add a second or more instance(s) of the driver without modification. Note: the load-balanced drivers must be hosted on separate DirXML servers in order to obtain the desired performance benefits.

The diagram below illustrates how the JMS Driver can be readily configured for automatic fail over and load balancing.



Vendor Configurations

General Driver configuration

- All Implementations of the JMS Driver require these files to be copied to the \Novell\NDS\Lib directory or equivalent location depending on the platform required.
    - JMSDriver.Jar
    - JMS.JAR
    - Vendor Specific JMS Libraries

Supported JMS Vendor Implementations.

- JMS
- Oracle Advanced Queuing
- IBUS Message Server
- TIBCOJMS
- TIBCO
- JBroker
- Open JMS

JMS

Summary

IBM WebsphereMQ (MQ Series) is the defining message queue product in its class and used in the great majority messaging implementations. WebsphereMQ implements full JMS compatibility in both standard and WebsphereMQAnyplace (lightweight) versions.

Company Web Site: www.ibm.com

MQ Series Specific Configurations.

Versions 5.2 and below need the MA88 Support Pack (Java Messaging System support).

The Support pack is available as a free download from
http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma88.html

Version 5.3 and higher you will need to select custom installation and select the option to install Support for JMS.

If you wish to Use topics you will the MA0C support pack and start the Strmbrk process. The support pack is available as a free download from
http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma0c.html

Vendor Specific Support Files
- Connector.jar
- Com.ibm.jms.jar
- Com.ibm.jmsbind.jar
- Com.ibm.mqjms.jar

Authentication Information.
The User id that will authentication to the JMS environment must be a member of the MQM group.

Windows Environment Configuration

Add the JMS\Java\Lib directory to the windows path environment variable.

Solaris Environment Configuration.

Add the following path to the java support files. These files are usually located in the following directories
- /opt/mqm/bin:
- /opt/mqm/lib:
- /opt/mqm/java/lib

This path needs to be added to the LD_LIBRARY_PATH in the NDSD startup script for the MQ driver to find the supporting c libraries.

** note you must also add make sure you export the LD_LIBRARY_PATH

JMS Remote Configuration

MQ Series Specific Configurations.

Remote connections require that the JMS Client must be installed.

Versions 5.2 and below need the MA88 Support Pack (Java Messaging System support). Later versions have integrated JMS support within the standard installation procedure.

For those that require it, the Support pack is available as a free download from http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma88.html

Version 5.3 and higher you will need to select custom installation and select the option to install Support for JMS.

If you wish to use topics you will the MA0C support pack and start the Strmbrk process. The support pack is available as a free download from http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma0c.html

Vendor Specific Support Files

- Connector.jar
- Com.ibm.jms.jar
- Com.ibm.jmsbind.jar
- Com.ibm.mqjms.jar

Authentication Information.
The User id that will authentication to the JMS environment. The user must be a member of the MQM group.

Windows Environment Configuration

Add the JMS\Java\Lib directory to the windows path environment variable.

Solaris Environment Configuration.

Add the following path to the java support files. These files are usually located in the following directories

- /opt/mqm/bin:
- /opt/mqm/lib:
- /opt/mqm/java/lib

Oracle Advanced Queuing (AQ)

Summary

Oracle AQ is the preferred approach used by Oracle to develop highly abstracted, efficient and safe application integration. AQ is integral to all Oracle databases from 8.04 onwards and supports the JMS standard.

Company Web Site: www.oracle.com

Oracle Specific Configurations.

Java Messaging System support is included in every version of Oracle from 8.04.

Supporting Files

- aqapi.jar (Advanced Queuing files)
- ojdbc14.jar (oracle thing jdbc thin driver support)

Authentication Information.

The User id that will authentication to the Oracle Advanced Queuing environment must be a member of the AQ Users Role or AQ Administrators Role.

Oracle Advanced Queuing Message Format

Some difficulty may be experienced when de-queuing a message generated by workflow and formatted by the queue handler. It is necessary with some versions of oracle to populate the agent section, header section, property section and the text section of the message. A JMS-120 is generated when the suggested sections were not populated appropriately. The error is as follows:

```
JMS-120: Dequeue failed
oracle.jms.AQjmsException: JMS-120: Dequeue failed
at oracle.jms.AQjmsError.throwEx(AQjmsError.java:233)
at oracle.jms.AQjmsConsumer.dequeue(AQjmsConsumer.java:1429)
at oracle.jms.AQjmsConsumer.receiveFromAQ(AQjmsConsumer.java:691)
at oracle.jms.AQjmsConsumer.receive(AQjmsConsumer.java:628)
```

An example of populating the message with each section defined correctly is as follows:

```
l_jms_text_message:= sys.AQ$_JMS_TEXT_MESSAGE ( sys.AQ$_JMS_HEADER
        (sys.AQ$_AGENT(' ', NULL, 0),
        NULL,
        'apps',
        NULL,
        NULL,
        NULL,
        sys.AQ$_JMS_USERPROPARRAY
          ( sys.AQ$_JMS_USERPROPERTY('jmsobject', 200, NULL, 12345, 23)
        )
       length(p_jms_text_message.text_vc), p_jms_text_message.text_vc, NULL
     );
```

Soft-wired iBUS

Summary

iBus is a highly robust and cost effective java messaging product. Unique features include both Internet (SSL) and Mobile device connectivity.

Company Web Site: www.soft-wired.com

iBUS Specific Configurations.
Special Vendor Installation Options

Supporting Files

- IBUSmsrvClt.jar

Authentication Information.

By default security is not implemented on the IBUS message server. If you wish to use security you must the security manager module in the administration tool. This module will allow you to assign ACL (Access Control Lists) to queues or topics.

Environment Settings

- Windows

  The IBUSSRV_HOME environment variable must be set before running the server scripts. The installer will have set the environment variable IBUSSRV_HOME to INSTALL_DIR\server. Use the *startserver.bat* batch file in the directory INSTALL_DIR\server\bin and the *stopserver.bat* batch file located in INSTALL_DIR\client\bin.

- Unix

  Set the environment variable IBUSSRV_HOME to INSTALL_DIR/server. To start the server, use the startserver.sh shell script in INSTALL_DIR/server/bin. To stop the server, use the *stopserver.sh* shell script in INSTALL_DIR/client/bin. This script is an iBus client and can be run from any machine.

TIBCO JMS Server

Summary

This message bus provides real-time connectivity to dozens of platforms and applica-tions.

Company Web Site: www.tibco.com

Tibco JMS Specific Configurations.

Vendor: TibcoJMS

Special Vendor Installation Options

Supporting Files

- TIBCOJMS.JAR

Authentication Information.

By default security is not implemented on the TIBCO JMS message server. If you wish to use security you must assign ACL (Access Control Lists) to queues or topics.

Environment Settings

None.

Spiritsoft SpriritWave TIBCO-JMS Bridge

Summary
This 3rd party product provides native connectivity to the TIBCO Rendezvous Message Bus without requiring the TIBCO JMS Server.

Company Web Site: www.spiritsoft.com

Special Vendor Installation

To properly register the Spiritsoft-TIBCO Bridge license file with the eDirectory instance you will need to copy the Properties directory to the: \NOVELL\NDS\jre\lib\ext Directory.

Supporting Files

- rvconfig.jar
- tibrvj.jar
- tibrvjsd.jar

Novell jBroker

Summary

This product provides a high-performance JMS-compliant message bus that integrates with all Novell exteNd technologies as well as all other J2EE technology.

Company Web Site: www.novell.com

Jbroker Specific Configurations.

Vendor: Jbroker

Special Vendor Installation Options

Supporting Files

- jbroker-mq.jar
- jbroker-rt.jar
- jbroker12.dll

Authentication Information.

By default security is not implemented on the IBUS message server. If you wish to use security you must assign ACL (Access Control Lists) to queues or topics.

Environment Settings

JBroker versions 2.1 and above

A search path must be set for the jbroker12.dll

Copy the Directories

- \ Jdk1.x\jre\lib\ext\classes
- \ Jdk1.x\jre\lib\ext\x86\

to:

\Novell\NDS\jre\lib\ext Directory

OpenJMS

Summary

Product Web Site: openjms.sourceforge.com

OpenJMS Specific Configurations.

Vendor: Open Source

Authentication Information.

By default security is not implemented on the OpenJMS message server. If you wish to use security you must assign ACL (Access Control Lists) to queues or topics.

Appendix A: JMS Overview

What Is JMS Messaging?

Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility: A messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages.

Messaging enables distributed communication that is *loosely coupled*. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only what message format and what destination to use. In this respect, messaging differs from tightly coupled technologies, such as Remote Method Invocation (RMI), which require an application to know a remote application's methods.

Messaging also differs from electronic mail (e-mail), which is a method of communication between people or between software applications and people. Messaging is used for communication between software applications or software components.

Message types

In JMS, a message is a Java object with 3 parts: a header, properties and a message body.

- Message header
- Message properties
- Message body

The following sections describe the three parts in detail.

The Message Header

Every JMS message includes message header fields that are always passed from producer to consumer. The purpose of the header fields is to convey extra information to the consumer outside the normal content of the message body. The JMS provider sets some of these fields automatically after a message is sent to the consumer, but the Message Producer has the opportunity to set some fields programmatically.

JMSCorrelationID

The JMSCorrelationID header field provides a way to correlate related messages. This is normally used for a request/response scenario. This can either be a vendor-specific ID, an application-specific string, or a provider-native byte value.

JMSExpiration or Time to Live

The JMSExpiration header field specifies the expiration or time-to-live value for a message. If the value is set to 0, the message will never expire. When a message does expire, the JMS provider typically will discard the message. Also, any persistent messages will be deleted based on expiration values.

JMSMessageID

The JMSMessageID header field contains a value that uniquely identifies each message sent by a provider. This value is set by the provider automatically and returned to the message producer when the send method completes. All JMSMessageID values must start with an ID: prefix.

JMSPriority

JMS defines 10 priority levels, 0 through 9. 0 is the lowest priority and level 9 is the highest. Levels 0–4 indicate a range of normal priorities, and levels 5–9 indicate a range of expedited priority. Priority level 4 is typically the default for a message producer.

Message Properties

The message property fields are similar to header fields described previously in the The Message Header section, except these fields are set exclusively by the sending application.

The Message Body

The message body carries free form application data, which can take several forms: text, serializable objects, byte streams, etc. The JMS API defines several message types (TextMessage,ByteMessage MapMessage, and others) and provides methods for delivering messages to and receiving messages from other applications.

JMS Messaging Models: Publish-and-Subscribe and Point-to-Point

JMS provides two types of messaging models: *publish-and-subscribe* and *point-to-point*. The JMS specification refers to these as *messaging domains*. In JMS terminology, publish-and-subscribe and point-to-point are frequently shortened to pub/sub and p2p (or PTP), respectively. This chapter uses both the long and short forms throughout.

In the simplest sense, publish-and-subscribe is intended for a one-to-many broadcast of messages, while point-to-point is intended for one-to-one delivery of messages

Publish-and-subscribe

In publish-and-subscribe messaging, one producer can send a message to many consumers through a virtual channel called a *topic*. Consumers can choose to *subscribe* to a topic. Any messages addressed to a topic are delivered to all the topic's consumers. Every consumer receives a copy of each message. The pub sub messaging model is by and large a push-based model, where messages are automatically broadcast to consumers without them having to request or poll the topic for new messages.

In the pub/sub messaging model, the producer sending the message is not dependent on the consumers receiving the message. Optionally, JMS clients that use pub/sub can establish durable subscriptions that allow consumers to disconnect and later reconnect and collect messages that were published while they were disconnected.

Point-to-point

The point-to-point messaging model allows JMS clients to send and receive messages both synchronously and asynchronously via virtual channels known as *queues*. The p2p messaging model has traditionally been a pull- or polling-based model, where messages are requested from the queue instead of being pushed to the client automatically. (The JMS specification does not specifically state how the p2p and pub/sub models must be implemented. Either one may use push or pull, but at least conceptually pub/sub is push and p2p is pull.)
A queue may have multiple receivers, but only one receiver may receive each message. The JMS provider will take care of doling out the messages among JMS clients, ensuring that each message is consumed by only one JMS client. The JMS specification does not dictate the rules for distributing messages among multiple receivers.

Appendix B: Supporting National Language Code Pages

The driver supports different code pages for national languages such as Japanese by setting the JMS Queue Manager CCSIS to the code page corresponding to the UTF-8 character format for the target language.

Steps to setting the Queue Manger CCSID

1. Execute the runmqsc program. This will bring up a DOS-like box
2. Enter the command: ALTER QMGR CCSID (1208)

Messages sent to the queue need to be in Unicode format. Information on the Unicode standard can be found at http://www.unicode.org/

*NOTE: The value 1208 represents the Japanese language UTF-8 code page. Substitute the correct UTF-8 code page ID for the required target language.

Appendix C Driver Parameters XML

```
<?xml version="1.0" encoding="UTF-8"?>
<driver-config name="JMS Driver">
        <driver-options>
                <qmgr display-name="Queue Manager">QM_appbox</qmgr>
                <msgbrk display-name="Message Broker"/>
                <mqchannel display-name="WebSphere MQ Channel"/>
                <sp display-name="Security Principal"/>
                <purl display-name="Provider Url">127.0.0.1</purl>
                <sc display-name="Security Credentials"/>
                <ss display-name="Security Protocol"/>
                <vb display-name="Verbose">TRUE</vb>
                <port display-name="OraclePort"/>
                <osid display-name="OracleSID"/>
                <ouser display-name="Oracle-Schema"/>
                <otable display-name="OracleQueueTable"/>
                <clientid display-name="JMS Client ID"/>
                <vendor display-name="Vendor">JBROKER</vendor>
        </driver-options>
        <subscriber-options>
                <qsend display-name="Queue Sender">subq</qsend>
                <tpub display-name="Topic Sender"/>
                <nonper display-name="Message persistence"/>
                <bytemessaging display-name="Byte Messaging"/>
                <jmsclient display-name="Client  Mode"/>
                <subqueryq display-name="Subscriber Query Queue"/>
                <subqueryreplyq display-name="Subscriber Query Reply Queue"/>
                <subquerytimeout display-name=" Subscriber Query Reply Timeout"/>
                <ttl display-name="JMS Time To Live"/>
                <pri display-name="JMS Priority"/>
        </subscriber-options>
        <publisher-options>
                <qrec display-name="Publisher Queue Receiver">pubq</qrec>
                <queryq display-name="Publisher Query Queue"/>
                <queryreplyq display-name="Publisher Query Reply Queue"/>
                <querytimeout display-name="Publisher Query Reply Timeout"/>
                <trec display-name="Topic Receiver"/>
                <headers display-name="Retrieve Headers"/>
                <jmsclient display-name="Client  Mode"/>
                <auditq display-name="Error Auditing Queue"/>
                <trackingq display-name="Transaction Tracking Queue"/>
                <correlationid display-name="Use Correlation ID'S"/>
        </publisher-options>
```

Appendix D Solutions

RACF & CICS Integration

Overview

The RACF & CICS integration solution is a mainframe-administrator friendly approach to integrating RACF and other host systems using CLIST files. A set of included ReXX scripts extract message containing information for the RACF, CICS or other databases on the host. The solution uses IBM JMS product to transport the information from the source (eDirectory) to the Mainframe and ReXX to execute. This is a very non-intrusive solution as it provides the mainframe administrators total control over the integration.

IBM Support Packs required:

- MA12 (Batch Trigger Monitor)
- MA18 (MQ Rexx Support)

Where to find these support packs:

Both these support pacs can be found on IBM's website under Category 2, AS-IS SupportPacs

- http://www-3.ibm.com/software/integration/support/supportpacs/

Supporting information:

How it works

The trigger monitor submits a batch job that invokes a batch TSO TMP where a Rexx exec program fetches messages from a queue.

Sample Rexx Script

```
/* REXX */
/* TRACE !RESULTS */

PARSE UPPER ARG ,
  USID "|" ,
  PSWD "|" ,
  UNAME "|" ,
  BG "|" ,
  DG "|" ,
  OG "|" ,
  TSOFLG "|" ,
  PRACNO "|" ,
  JOBF "|" ,
  REG "|" ,
  LOC "|" ,
  VMID "|" ,
  EMPTYPE "|",
  HRT "|"

 MAXCC = 0
 UEXIST = BLANK

 SAY "*** START PROCESSING USID="USID "***"
 SAY "UNAME="UNAME "VMID="VMID
 SAY "BG="BG "DG="DG "OG="OG
 SAY "TSOFLG="TSOFLG "PRACNO="PRACNO "JOBF="JOBF
 SAY "REG="REG "LOC="LOC "EMPTYPE="EMPTYPE "HRT="HRT

 IF HRT = TER THEN DO
   MAXCC = 0
   SAY "*** NOT PROCESSED: TERMINATION ***"
   EXIT MAXCC
   END

 IF USID = " THEN DO
   MAXCC = 0
   SAY "*** PROBLEM: NO USERID ***"
   EXIT MAXCC
   END

 AP = POS("'",UNAME)
 IF AP > 0 THEN UNAME = INSERT("'",UNAME,AP)

 X = OUTTRAP('LUOUT',0)
"LU" USID
 LURC = RC
 X = OUTTRAP('OFF')

 SELECT
   WHEN LURC = 0 THEN DO
    "CO" USID "GROUP("DG")"
    "ALU" USID "RESUME"
    "RE" USID "GROUP(@ISTERM)"
     UEXIST = Y
     IF MAXCC < 4 THEN MAXCC = 4
     SAY "*** NOTICE:" USID "EXISTED ***"
     END
   WHEN LURC = 4 THEN DO
    "AU" USID "OW("OG") DFLTGRP("DG")"
     AURC = RC
     UEXIST = N
     END
   OTHERWISE SAY "*** PROBLEM: LU ***"
 END

 IF HRT = REH THEN "CO" USID "GROUP(@ISTERMX) OW(ISSD01)"

 IF UEXIST = N THEN ,
   IF AURC ¬= 0 THEN DO
     IF MAXCC < AURC THEN MAXCC = AURC
     SAY "*** PROBLEM: AU ***"
     END

"ALU" USID "OW("OG") DFLTGRP("DG") PASSWORD("PSWD")" ,
   "NAME('"UNAME"') DATA('"VMID"') CICS(OPIDENT(HAL))"
 ALRC = RC

 IF ALRC ¬= 0 THEN DO
   IF MAXCC < ALRC THEN MAXCC = ALRC
   SAY "*** PROBLEM: ALU ***"
   IF MAXCC > 4 THEN EXIT MAXCC
   END

 SELECT
   WHEN SUBSTR(USID,1,2) = @N THEN DO
     UUID = 80||SUBSTR(USID,3,4)
    "ALU" USID "OMVS(UID("UUID") HOME(/) PROGRAM(/bin/echo))"
     END
   OTHERWISE DO
     UUID = 1||SUBSTR(USID,2,5)
    "ALU" USID "OMVS(UID("UUID") HOME(/) PROGRAM(/bin/echo))"
     END
 END

 IF TSOFLG ¬= Y THEN EXIT MAXCC

"CO" USID "GROUP("BG") OW("OG")"

 SELECT
   WHEN OG = @TBADM01 THEN DO
     PRCD = LOGONU
     CMND = "EX 'BAS0000.P.TOOLS.CLIST(LOGON)'"
     END
   OTHERWISE DO
     PRCD = LOGONI
     CMND =
     END
 END

"ALU" USID "TSO(ACCTNUM(00000-00521109-001-51-238S)" ,
  "SIZE(8192) MAXSIZE(8192) JOBCLASS(A) MSGCLASS(S) SYSOUTCLASS(S)" ,
  "DEST(LOCAL) UNIT(SYSDA) PROC("PRCD") COMMAND('"CMND"'))"

 V1 = RANDOM(0,7)
 ACCNT = 0

 X = OUTTRAP('LCAT',0)
"LISTCAT ENTRIES('"USID"')"
 LCRC = RC
 X = OUTTRAP('OFF')

 SELECT
   WHEN LCRC = 0 THEN DO
     IF MAXCC < 4 THEN MAXCC = 4
     SAY "*** NOTICE: CAT ENTRY EXISTED ***"
     END
   WHEN LCRC = 4 THEN DO
     DO UNTIL ACRC = 0 | ACCNT = 8
       TUCAT = SYS1.UCATTSO||V1
       IF TUCAT = SYS1.UCATTSO0 THEN TUCAT = SYS1.UCATTSO
      "DEFINE ALIAS (NAME('"USID"') RELATE('"TUCAT"'))"
       ACRC = RC
       ACCNT = ACCNT + 1
       V1 = V1 + 1
       IF V1 = 8 THEN V1 = 0
     END
     IF ACRC ¬= 0 THEN DO
       IF MAXCC < ACRC THEN MAXCC = ACRC
       SAY "*** PROBLEM: DEFINE ALIAS ***"
       EXIT MAXCC
       END
     END
   WHEN LCRC > 4 THEN DO
     IF MAXCC < LCRC THEN MAXCC = LCRC
     SAY "*** PROBLEM: CATALOG ENTRY ***"
     END
 END

"ALLOC DA('"USID".ISPPROF') NEW SPACE(2,1) DIR(35) TRACKS" ,
  "DSORG(PO) RECFM(F B) LRECL(80) BLKSIZE(6160)"
```

```
"FREE DA('"USID".ISPPROF')"
"ADDSD '"USID".*' OWNER("USID") UACC(NONE) AUDIT(NONE)"
"PE '"USID".*' ID("BG") ACC(READ) GEN"
"ADDSD '"USID".SYSOUT.*' OWNER("USID") UACC(NONE) AUDIT(NONE)"
"PE '"USID".SYSOUT.*' ID("BG") ACC(READ) GEN"

 SELECT
   WHEN PRACNO = 11 THEN NOP
   WHEN PRACNO = 14 THEN NOP
   WHEN PRACNO = 15 THEN NOP
   WHEN PRACNO = 17 THEN NOP
   WHEN PRACNO = 19 THEN ,
     TBIDFLAG = YES
   WHEN PRACNO = 20 THEN NOP
   WHEN PRACNO = 24 THEN ,
     TBIDFLAG = YES
   WHEN PRACNO = 27 THEN NOP
   WHEN PRACNO = 32 THEN NOP
   WHEN PRACNO = 40 THEN NOP
   WHEN PRACNO = 41 THEN NOP
   WHEN PRACNO = 43 THEN NOP
   WHEN PRACNO = 44 THEN NOP
   WHEN PRACNO = 64 THEN ,
    "PE '"USID".SYSOUT.*' ID(@HETEC01) ACC(ALTER) GEN"
   WHEN PRACNO = 67 THEN NOP
   OTHERWISE
 END

 IF TBIDFLAG = YES THEN DO
   USIB = USID||'B'
   X = OUTTRAP('LUBOUT',0)
  "LU" USIB
   LUBRC = RC
   X = OUTTRAP('OFF')
   SELECT
     WHEN LUBRC = 0 THEN DO
      "ALU" USIB "RESUME"
      "RE" USIB "GROUP(@ISTERM)"
       IF MAXCC < 4 THEN MAXCC = 4
       SAY "*** NOTICE:" USIB "EXISTED ***"
       EXIT MAXCC
       END
     WHEN LUBRC = 4 THEN DO
      "AU" USIB "OW(@BAADM01) DFLTGRP(@TBHLD) PASSWORD("PSWD")" ,
        "NAME('"UNAME"') DATA('TBA B ID') CICS(OPIDENT(HAL))" ,
        "TSO(ACCTNUM(00000-00521109-001-51-238S) PROC(LOGONB)" ,
        "SIZE(2048) MAXSIZE(8192) JOBCLASS(A) MSGCLASS(S) " ,
        "SYSOUTCLASS(S) DEST(LOCAL) UNIT(SYSDA))"
       ABRC = RC
       IF ABRC ¬= 0 THEN DO
         IF MAXCC < ABRC THEN MAXCC = ABRC
         SAY "*** PROBLEM: AU BID ***"
         EXIT MAXCC
         END
       ELSE DO
        "CONNECT" USIB "GROUP(@ISTBAB) OW(@ISADM01)"
         IF TUCAT = TUCAT THEN TUCAT = SYS1.UCATTSO
        "DEFINE ALIAS (NAME('"USIB"') RELATE('"TUCAT"'))"
        "ALLOC DA('"USIB".ISPPROF') NEW   SPACE(2,1) DIR(35) TRACKS " ,
          "DSORG(PO) RECFM(F B) LRECL(80) BLKSIZE(6160)"
        "FREE DA('"USIB".ISPPROF')"
        "ADDSD '"USIB".*' OWNER("USIB") UACC(NONE) AUDIT(NONE)"
        "PE '"USID".*' ID("USIB") ACCESS(ALTER) GEN"
        "ADDSD '"USIB".P.AQUA.*' OWNER("USIB") UACC(NONE) AUDIT(NONE)"
        "PE '"USIB".P.AQUA.*' ID(@ISPBIDS) ACCESS(READ)"
        "PE '"USIB".P.AQUA.*' ID(@DBDBA01) ACCESS(READ)"
       "ADDSD '"USID".P.AQUA.*' OWNER("USID") UACC(NONE)" ,
          "FROM('"USID".*')"
        "PE '"USID".P.AQUA.*' ID(@DBDBA01) ACCESS(READ)"
        "RACLINK ID("USID") DEFINE(HALINMA1."USIB") PEER(PWSYNC)"
         END
       END
     OTHERWISE SAY "*** PROBLEM: LU B ID ***"
   END
 END

 RETURN MAXCC
```

Oracle Integration

Company Web Site: www.oracle.com

Overview

Oracle provides a set of JMS interfaces and associated semantics called Advanced Queue (AQ) that define how a JMS client accesses can the facilities of Oracle using industry-standard Java messaging.

Oracle JMS supports the standard JMS interfaces and has extensions to support the AQ administrative operations and other AQ features that are not a part of the JMS standard.

The Oracle Advanced Queuing interfaces allows the developer to quickly and safely integrate with a number of Oracle applications through messaging, as opposed to directly updating database tables. This allows API's developed in PLSQL and Java to prepare, validate, execute, roll-back and post-process all DirXML updates to Oracle to ensure complete data integrity. Applications that can be integrated in this fashion include:

- Oracle HR
- Oracle Financials
- Oracle E-Business Suite
- Etc…

Oracle Workflow

Oracle Workflow is an increasingly popular toolset that automates and streamlines business processes both internal and external to the enterprise. It supports traditional applications-based workflows as well as Identity Management integration workflows.

The Workflow Engine embedded in the Oracle database server monitors workflow states and coordinates the routing of activities for a process. Changes in workflow state, such as the completion of workflow activities, are signaled to the engine via a Java API. Based on flexibly defined workflow rules, the engine determines which activities are eligible to run and the schedule that they use.

Oracle Workflow leverages Oracle Advanced Queuing and enables DirXML application integration at the business process level. Business event messages from DirXML can be placed on or received from Oracle Advanced Queues and consumed by Oracle Workflow. Support is also for HTTP and HTTPS communications protocols.

Oracle Workflow Business Event System

A business event is triggered in an application whenever something of significance happens. An example of a business event is the hiring of an employee or the creation of a customer. The Business Event System consists of the Event Manager, which lets you register subscriptions to significant events, and event activities, which let you model business events within workflow processes.

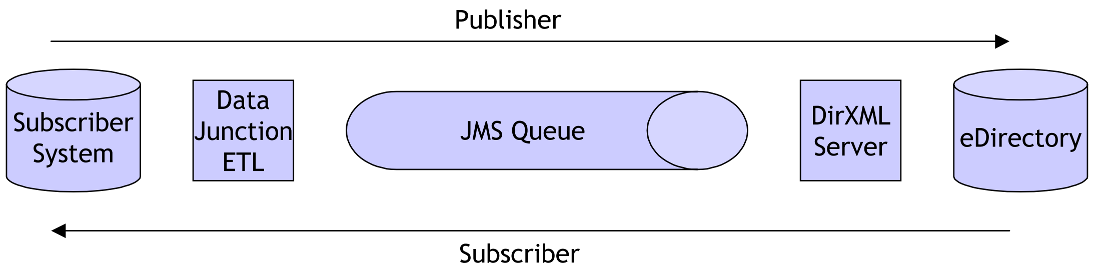
When a local event occurs, the subscribing code is executed in the same transaction as the code that raised the event, unless the subscriptions are deferred. Subscription processing can include executing custom code on the event information, sending event information to a workflow process, and sending event information to other queues or systems.

Data Junction Integration

Company Web Site: www.datajunction.com

Overview

Data Junction is an award-winning Extraction, Loading & Transformation (ETL) product that provides powerful data manipulation and extensive connectivity tools. When configured with the DirXML Driver for JMS/MQ, it enables eDirectory to connect to hundreds any application, platforms and data formats.



The Data Junction product suite provides a Visual Map Designer, Process Designer, Structure Schema Designer, Document Schema Designer and Extract Schema Designer.

In addition, the Data Junction suite provides includes highly specialized adapters for a wide variety of application services, all of which will integrate seamlessly with DirXML using the JMS driver. These include the following:

Dialog Adaptors:

- Biographical
- Business & News
- Commerce Business Daily
- Company Directory
- Federal Registry
- Market Research
- Patents
- Sci/Tech & Bio/Med
- Trade Names & Products
- Trademarks

Dodge Adaptors

- Bidders
- Firms
- Firms & Bidders

- Projects

Application Adaptors:

- AccPac
- EDI (X12)
- EDI (EDIFACT)
- HIPAA
- HL7
- SAP (IDoc)I
- SWIFT (Target only)
- STN Chemical
- Dow Jones
- Internet Email
- Reuters
- Xerox

HIPAA Adaptors

- HIPAA Adapter users: Refer to HIPAA for a detailed list of instructions on how to connect to HIPAA Health Claim Source Files.

General Adaptors

- Access 2000
- Access 97
- Access XP
- AccountMate
- ACT! for Windows
- Acucobol (ODBC 3.x)
- Adabas (NatQuery)
- Alpha Four
- Apache Common Logfile Format
- ASCII (Delimited)
- ASCII (Fixed)
- BAF
- Binary Line Sequential
- Binary
- BizTalk XML
- Btrieve v6
- Btrieve v7
- C-ISAM
- C-tree 4.3
- C-tree Plus
- Cambio
- Clarion
- Clipper
- Cloudscape(ODBC)
- Cobol Flat File
- Common Logfile Format Webserver
- Content Extractor
- Data Junction Log File
- DataEase
- DataFlex (ODBC 3.x)
- dBASE II
- dBASE III+
- dBASE IV
- dBASE V (IDAPI)
- DIALOG Biographical
- DIALOG Business and News
- DIALOG Commerce Business Daily

- DIALOG Company Directory
- DIALOG Federal Register
- DIALOG Market Research
- DIALOG Patents
- DIALOG Sci_Tech and Bio_Med
- DIALOG Trade Names and Products
- DIALOG Trademarks
- dialog.djx
- DIF
- EDI (X12)
- edi4.djx
- EDIFACT
- Enable
- Excel 2000
- Excel 95
- Excel 97
- Excel v2
- Excel v3
- Excel v4
- Excel v5
- Excel XP
- eXcelon 2.x
- eXcelon XIS 3.0
- Extractor
- FIX
- FIXML
- Folio Flat File
- Foxbase+
- FoxPro
- Fujitsu Cobol
- GoldMine Import File (dbf)
- GoldMine
- Great Plains DOS (Btrieve)
- Great Plains Unix_Mac (C-tree)
- HCFA1500 - NSFA
- HIPAA
- Hitachi HiRDB (ODBC)
- HTML
- IBM DB2 7.2 Universal Database Multimode
- IBM DB2 Loader
- IBM DB2 UDB Mass Insert
- IBM DB2 Universal Database Multimode
- IDAPI
- Informix (ODBC 3.x)r
- Informix DB Loader
- Informix-Online DS Multimode

- Informix_SE
- Ingres (ODBC 3.x)
- Interbase (IDAPI)
- Join engine
- LDAP
- LDIF
- Lotus 123 r1A
- Lotus 123 r2
- Lotus 123 r3
- Lotus 123 r4
- Lotus Notes 5
- Lotus Notes Structured Text
- Lotus Notes Structured Text
- Lotus Notes
- Macola Acct (Btrieve)
- Magic PC
- MAILER'S+4 (dBase)
- Micro Focus COBOL
- Microsoft COBOL
- Microsoft IIS Extended Logfile Format
- MUMPS (ODBC)
- Navision Financials (ODBC 3.x)
- NonStop SQL_MX (ODBC)
- ODBC 3.5 MultiMode
- ODBC 3.5
- ODBC 3.5
- ODBC 3.x Mass Insert
- ODBC 3.x MultiMode
- ODBC 3.x
- ODBC 3.x
- Oracle 7.x
- Oracle 7.x
- Oracle 8.x Multimode
- Oracle 8.x
- Oracle 8.x
- Oracle 9i Multimode
- Oracle 9i
- Oracle 9i
- Paradox v5 (IDAPI)
- PayChex (DJF) Import
- Personal Librarian
- Pervasive.SQL
- Platinum Acct (Btrieve)
- PostgreSQL
- Progress (ODBC 3.x)
- Quattro Pro Windows v5

- RBase (ODBC 3.x)
- RealWorld Acct (MFCOBOL)
- Red Brick
- Remedy ARS
- Report Reader
- Rich Text Format
- salesforce.com
- SAP (IDoc)
- SAS Transport Format
- SBT Acct (FoxPro)
- Scalable SQL
- Sequential Binary
- SGML
- Solomon Acct (Btrieve)
- SPLUS
- SPSS
- SQL Script
- SQL Server 2000
- SQL Server 6.x
- SQL Server 7
- SQL Server BCP
- SQL Server Mass Insert
- SQLBase 6.X
- SQLBase
- Statistica
- SWIFT
- Sybase Adaptive Server 11.x
- Sybase Adaptive Server 12.x
- Sybase BCP
- Sybase SQL Anywhere 6
- Sybase SQL Anywhere
- Sybase SQL Server Mass Insert
- Sybase SQL Server System 11 Multimode
- SYSTAT
- Tape Drive Sequential
- Teradata (Fastload)
- Text (Delimited - EDI)
- Text (Delimited - EDI)
- Text (Delimited - EDIFACT)
- Text (Delimited - EDIFACT)
- Text (Delimited - HL7)
- TRADACOMS
- UB92 - NSF
- Unicode (Delimited)
- Unicode (Fixed)
- USMARC

- Variable Sequential (MVS)
- Velocis (ODBC 3.x)
- Visual dBASE 5.5
- Visual FoxPro
- Watcom SQL v5
- WordPerfect 6.0 (Mail Merge)
- XDB (ODBC 3.x)
- XML