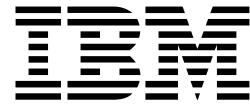


WebSphere® Data Interchange for
Multiplatforms



User's Guide

Version 3.1

WebSphere® Data Interchange for
Multiplatforms



User's Guide

Version 3.1

Note: Before using this information and the products it supports, read the information in “Notices” on page 59.

First Edition (April 2002)

This edition applies to Version 3.1 of IBM WebSphere Data Interchange for Multiplatforms (product number 5724-C50) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1998, 2002. All rights reserved.

US Government Users Restricted Rights — Use, duplication, or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book	v	Triggering from an MQSeries Queue.	21
Who this book is for.	v	Adapter user exits	23
What you need to know	v	Calling from a C++ program.	25
Related books	v	Elements of the C++ API	25
Glossary.	vi	WebSphere Data Interchange API	
How to send your comments.	vi	example.	33
Introduction	1	Message Content Descriptor	
WebSphere Data Interchange.	1	profiles	39
Flexible setup and administration	1	Setup overview	39
Superior translation capability	2	Creating MCD profiles.	41
Versatile communications.	2	 	
Extensive reporting and auditing.	2	Service profiles.	43
EDI standards support.	3	Substitution keywords	43
Additional features.	3	Setup overview	44
 		Creating Service profiles	45
Installation	5	 	
WebSphere Data Interchange Server	5	C++ and Java API return codes	51
Installing WebSphere Data Interchange		 	
Server	6	Mapping the MQRFH2 header to	
WebSphere Data Interchange Client.	11	the JMS API.	57
Installing WebSphere Data Interchange		 	
Client	11	Notices	59
Setting up connections to server		Trademarks	61
databases	14	 	
 		Index	63
Running the WebSphere Data			
Interchange Server	19		
Running from the command line	19		

Contents

About this book

This guide contains information and instructions for installing and using IBM® WebSphere® Data Interchange for Multiplatforms Version 3.1.

Note: The WebSphere Data Interchange for Multiplatforms Version 3.1 component is based on the DataInterchange Version 3.1 product.

Who this book is for

This document is intended for the developer designing a solution and for the system administrator responsible for installing and managing WebSphere Data Interchange.

What you need to know

Developers and those who run solutions should have an understanding of data transformation and mapping.

Related books

The DataInterchange Version 3.1 library includes the books listed below. These books may be used for reference.

All the DataInterchange books are available at <http://www.ibm.com/websphere/datainterchange>.

- *DataInterchange Client User's Guide* (SB34-2010)
This guide describes how to use, interface, create, and maintain DataInterchange profiles, maps, data formats, and EDI standards.
- *DataInterchange Administrator's Guide* (SB34-2002)
This guide describes the administrator activities for the product, including defining profiles, defining networks, and authorizing users. The audience for the book is the application programmer or data processing analyst.

About this book

- *DataInterchange Messages and Codes* (SB34-2000)
This book provides all logged and displayed messages, as well as information to aid users and support personnel in identifying problems. This reference guide is for the EDI administrator, the system programmer, and the database administrator.
- *DataInterchange Programmer's Reference* (SB34-2001)
This book provides information for the DataInterchange application programmer. It describes general-use programming and provides reference information for developing application programs that use DataInterchange.

Other related documents include:

- *DataInterchange V3.1 XML Technical Implementation Guide* - available at <http://www.ibm.com/websphere/datainterchange>.

Glossary

For glossary definition terms and abbreviations used in WebSphere Data Interchange, refer to the glossary in the *DataInterchange Client User's Guide* (SB34-2010).

How to send your comments

IBM welcomes your comments. You can send your comments electronically to the network ID listed below. Be sure to include your entire network address if you wish a reply.

Mail to: emd19@us.ibm.com

Chapter 1. Introduction

IBM WebSphere® Data Interchange for Multiplatforms Version 3.1 is a robust data translation and transaction management solution that can scale with Electronic Data Interchange (EDI) to provide improved back-office reporting.

Before you can use WebSphere Data Interchange for Multiplatforms to translate data, or to send or receive transactions, messages, or files, you must define certain information. This information describes how your system sends and receives data, how data is formatted in your application files and mapped to a standard, to whom you send data and from whom you receive data, and other pertinent information.

This product is provided on two CD-ROMs, one for the Server and one for the Client.

WebSphere Data Interchange

The WebSphere Data Interchange licensed product is a single application that reformats data for electronic transmission, and includes the following features:

- Flexible setup and administration
- Superior translation capability
- Versatile communications
- Extensive reporting and auditing
- EDI standards support
- Additional features

Flexible setup and administration

- Online customization of EDI standards, maps, and trading partner relationships
- Mapping designed to support:
 - Literals/constants
 - Accumulators, arithmetic and logical operations
 - Qualified loop and element mapping

Introduction

- Hierarchical loop mapping
- Envelope field mapping
- User exits at the field level
- User-defined translation and validation tables
- Boolean logic
- Maps that can be used by one or more trading partners
- Export/import functions to move data between test and production systems

Superior translation capability

- Syntax checking
- Test and production support
- Ability to translate and envelope separately
- Flexible command language interface
- Interactive, batch, event-driven, and real-time processing
- Automatic generation of functional acknowledgments

Versatile communications

- Support for networks and direct connections to trading partners
- Ability to resend individual transactions or entire envelopes
- Support for MQSeries® message queues as a means of sending and receiving messages

Extensive reporting and auditing

- Reporting of trading partner relationships, including what transaction sets are being used, and when the last communication occurred with a trading partner, and others
- Reporting of envelope and transaction status for both online and batch processing
- Providing exception reporting
- Setting acceptable error levels for the trading partner/map combination

- Reporting of SAP status for online and batch processing
- Providing optional audit log with archive recovery capability

EDI standards support

- Multiple EDI standards, including EDIFACT, X12, UCS, VICS, and Rail
- Multiple versions and releases of EDI standards
- Electronic distribution to speed delivery of new EDI standards
- Ability to migrate a map from one version or release of a standard to another, or from one transaction to another
- Online creation and customization of EDI standards
- Full EDI standards compliance checking (user option)

Additional features

- High throughput and performance
- Support for concurrent users and applications
- Support for shared trading partner profiles (minimal trading partners)
- Ability to process in multiple environments
- Support for encryption and authentication
- Application program interface (API) to integrate directly with your application
- Java program interface to integrate with MQSeries Adapter Offering (MQAO) -based applications
- Graphical User Interface (GUI) to simplify the management of profiles, EDI standards, data formats, and maps

Introduction

Chapter 2. Installation

This chapter provides instructions for the installation of WebSphere Data Interchange for Multiplatforms Version 3.1 Server and Client.

WebSphere Data Interchange Server

Before you install WebSphere Data Interchange for Multiplatforms Server, ensure that you have the required hardware and software.

Hardware requirements

Your hardware must meet the following minimum standards:

- Server capable of running the AIX® or Microsoft® Windows 2000 operating system
- CD-ROM drive for installing the distributed material
- Required network hardware and communication connection

Additional hardware requirements for Microsoft Windows 2000:

- Intel Pentium III processor at 933 Mhz or faster
- 1024 MB of memory
- Storage device with a minimum of 8 GB free space

Software requirements

Your software must meet the following minimum standards:

- One of the following operating systems for the Server:
 - AIX V4.3.3
 - Microsoft Windows 2000
- The following server databases and tools:
 - Administrative database supplied with the product
 - DB2 Universal Database Workgroup Edition V7.2 with open database connectivity (ODBC) (provided with the product)

Installation

- Optional operating products:
 - MQSeries® V5.2
 - Expedite for AIX V4.5 or Expedite for Windows V4.6

Installing WebSphere Data Interchange Server

WebSphere Data Interchange Server has an InstallShield Wizard that guides you through the installation process for AIX or Microsoft Windows 2000 installations.

AIX

To use the InstallShield Wizard, you must be logged in as the root user. If you are running from a remote terminal, you must be using X-Windows and your DISPLAY environment variable must be set to your X-Server IP address.

By default, WebSphere Data Interchange Server is installed to the `/usr/wdi/DIv3.1` directory. You must have at least 50 MB of free space on this file system.

1. If you are installing from a CD-ROM, insert the WebSphere Data Interchange Server CD and mount the CD-ROM drive.
 - a. Change to the directory containing the `wdi.aix` executable.
 - b. Run the `wdi.aix` executable to start the InstallShield Wizard.

The Welcome screen opens as the InstallShield Wizard prepares to install WebSphere Data Interchange Server.

2. Click **Next**.

The license agreement opens.
3. Click the appropriate button to accept the terms of the license agreement and to indicate you have read the notice and agree to its terms.
4. Click **Next**.

The Installer dialog box displays a listing of the installation directory and the total size requirement.
5. Click **Next**.

The InstallShield Wizard begins copying program files. To stop this process at any point, click **Cancel**.

The screen displays the successful installation message.

6. Click **Finish**.

The installation of the files is complete.

The following directories are created within the installation directory.

bin	runtime/edi
bind	runtime/eex
ddl	runtime/fak
deinstl	runtime/prt
hlp	runtime/qry
include	runtime/rcv
ixf	runtime/rpt
runtime	runtime/trk
runtime/adf	runtime/wrk
runtime/aex	runtime/xex
runtime/dicmd	runtime/xml
runtime/dicts	samples
runtime/dtlds	_uninst

Microsoft Windows 2000

To use the InstallShield Wizard, you must be logged in as an administrator.

By default, WebSphere Data Interchange Server is installed to the **C:\Program Files\IBM\WDI Server V3.1** directory. You must have at least 70 MB of free space on this file system.

1. Insert the WebSphere Data Interchange Server CD into the CD-ROM drive.
 - a. On the menu bar, click **Start > Run**.
 - b. Find the directory containing the wdi.exe executable.
 - c. Run the wdi.exe executable to start the InstallShield Wizard.

The Welcome screen opens as the InstallShield Wizard prepares to install WebSphere Data Interchange Server.

2. Click **Next**.

The license agreement opens.

3. Click the appropriate button to accept the terms of the license agreement and to indicate you have read the notice and agree to its terms.

Installation

4. Click **Next**.

The Installer dialog box displays a listing of the installation directory and the total size requirement.

5. Click **Next**.

The InstallShield Wizard begins copying program files. To stop this process at any point, click **Cancel**.

The screen displays the successful installation message.

6. Click **Finish**.

The installation of the files is complete.

The following directories are created within the installation directory.

bin	runtime\ees
bind	runtime\fak
ddl	runtime\prt
hlp	runtime\qry
include	runtime\rcv
ixf	runtime\rpt
runtime	runtime\trk
runtime\adf	runtime\wrk
runtime\aex	runtime\xex
runtime\dicmd	runtime\xml
runtime\dicts	samples
runtime\dtids	_uninst
runtime\edi	

Setting up the WebSphere Data Interchange databases

Perform the following steps as a user with DB2 administrator authority.

1. If you are installing on Windows, using your DB2 administrator user ID, select **Start > Programs > IBM DB2 > Command Window** to open the DB2 Command window. If you are installing on AIX, log in as a user with administrator authority.

The remaining database setup steps use this command window or login session.

2. Change the directory to the **ddl** directory under the installation directory.

- a. Issue these commands:

```
db2 create db ediec31e
db2 create db edict31e
```

This process creates the databases. When this process has successfully completed, the databases have been built.

- b. After the databases have been built, issue these commands:

```
altrec31
altrct31
```

This process alters some of the default parameters related to log file size and to the number of primary and secondary logs.

- c. Change to the DB2 directory which contains the bind files for the DB2 utilities. If you are installing on Windows, this directory typically has a name similar to **C:\Program Files\SQLLIB\bnd**. If you are installing on AIX, this directory typically has a name similar to **/u/<db2 instance>/sqllib/bnd**, where **<db2 instance>** is the ID of the instance owner.

Note: For AIX installations, you may need to specify a different file for the messages (for example **/tmp/bind.msg**) if you do not have write authority to the current directory.

- d. Issue these commands:

```
db2 connect to ediec31e
db2 bind @db2ubind.lst messages bind.msg grant public
db2 bind @db2cli.lst messages clibind.msg grant public
db2 connect reset
```

```
db2 connect to edict31e
db2 bind @db2ubind.lst messages bind.msg grant public
db2 bind @db2cli.lst messages clibind.msg grant public
db2 connect reset
```

- e. Change the directory to the **ddl** directory under the installation directory.

Installation

3. Issue these commands:
db2 -tf ediec31.ddl -l ec31.log
db2 -tf edict31.ddl -l ct31.log

This process creates the WebSphere Data Interchange for Multiplatforms tables, indexes, views, and so on.

4. Issue these commands:
db2 -tf grntec31.ddl -l grntec31.log
db2 -tf grntct31.ddl -l grntct31.log

The GRANT statements necessary for WebSphere Data Interchange Client access to the newly created tables are issued. The default is to issue GRANTS to public. You may wish to change public to specific user IDs or a group of authorized users.

5. Change the directory to the **ixf** directory under the installation directory.
6. Issue the following commands in sequence:
 - a. loadec31
 - b. loadct31

This process loads initial data into the DB2 tables. The loading may generate warnings, which can be ignored.

7. Change the directory to the **bind** directory under the installation directory.
8. Issue this command:
db2 -tf bindgrnt.fil -l bind.log

This process BINDs the WebSphere Data Interchange DB2 packages and GRANTS execute authority to public.

The set up of the WebSphere Data Interchange databases is complete.

WebSphere Data Interchange Client

WebSphere Data Interchange for Multiplatforms Client has an Install Wizard that guides you through installation. As with any installation, you should begin by closing any applications you have running. Make sure you have enough space on the hard drive on which you are installing the application, EDI standards, and data files. The minimum recommended space is 128 MB.

Note: If you want to run the Client from a local area network (LAN), see the readme file included with the product.

Before you install WebSphere Data Interchange Client, ensure that you have the required hardware and software.

Hardware requirements

- Intel Pentium PC
- CD-ROM drive for installing the distributed material
- 128 MB RAM

Software requirements

- One of the following operating systems:
 - Microsoft Windows® 95
 - Microsoft Windows® 98
 - Microsoft Windows® 2000
- DB2 Connect Personal Edition V7.2 (provided with the product)

Other requirement

- WebSphere Data Interchange Server

Installing WebSphere Data Interchange Client

1. Insert the WebSphere Data Interchange Client Installation CD ROM. The installation process should start automatically. If it does not, go to step a.
 - a. On the menu, click **Start > Run**.
The Run dialog box opens with the cursor in the Open field.

Installation

- b. In the Open field, type `x:\client\WDIClient V3.1.exe`, and click **OK**. *x* indicates your CD ROM drive.

The Welcome Screen opens as the Install Wizard prepares to install WebSphere Data Interchange Client.

2. Click **Next**.

The screen displays the copyright notice.

3. Click the appropriate button to accept the terms of the license agreement and to indicate you have read the notice and agree to its terms.

If this is the first time you are installing WebSphere Data Interchange Client, the Choose Destination Location dialog box opens. Select your installation location by clicking **Browse** and choosing the appropriate drive and directory. The drive can be selected from the drop-down list at the bottom of the dialog box. The default destination directory is **C:\Program Files\IBM\WDIClient V3.1**.

Note: Install WebSphere Data Interchange Client in a different directory than previous WebSphere Data Interchange Client versions.

4. Click **Next**.

The Setup Type dialog box opens. Choose the type of setup you want from the following choices:

- **Typical**

Select this option the first time you install WebSphere Data Interchange Client. This option installs all the common options and creates the databases for the software.

Attention: If you are reinstalling WebSphere Data Interchange Client and you select this option, you receive a warning that Install will overwrite your databases. Install will only overwrite the default 3.1 database files installed through a previous 3.1 install. This option will not overwrite databases installed on a user's database system, such as DB2.

- **Custom**

Select this option when you want to choose the options to install. This option is recommended for advanced users. If you are reinstalling the WebSphere Data Interchange Client, you should use the Custom setup to avoid overwriting your databases and drivers with the defaults.

5. Choose a Setup Type by clicking the appropriate option button. The default Setup type is Typical.

6. Click **Next**.

If you chose a Typical setup, go to the next step.

If you chose a Custom setup, the Select Components dialog box opens. You can choose whether to install:

- Program Files
- Crystal Report Files
- Database Files

To select a component, click the check box next to it.

7. Click **Next**.

The Installer dialog box displays a listing of the selected features and the total size required to install them.

8. Click **Next**.

The Install Wizard begins copying program files. To stop this process at any point, click **Cancel**.

The screen displays the successful installation message.

9. Click **Next**.

The readme file displays.

10. Click **Finish**.

Select **WDI Client** from the program folder specified during installation to start WebSphere Data Interchange Client.

Note: If you are upgrading WebSphere Data Interchange Client from a previous release, see the *DataInterchange Client User's Guide* for more information.

Setting up connections to server databases

The following instructions assume that CustTime and RunTime databases for WebSphere Data Interchange Server have been installed successfully and that you are authorized to access those databases. Do the following tasks in the order they are given.

1. Install DB2 Connect Personal Edition V7.2

- a. Insert the DB2 Connect Personal Edition CD that is included in the package.

The setup program should autostart; however, if it does not start, invoke setup.exe from the DB2 Connect Personal Edition CD.

- b. On the Installation dialog box, click **Install** and follow the prompts.

- 1) On the Select Products dialog box, select **DB2 Connect Personal Edition**. The other products are not needed.

- 2) On the Select Installation Type dialog box, select **Typical**.

- 3) On the Choose Destination Location dialog box, accept the default directory or choose an alternate installation destination.

- 4) On the Configure NetBIOS dialog box, accept the default.

- 5) On the Enter Username and Password for Control Center Server dialog box, type your username and password. If an error occurs due to authority, you can usually ignore it and continue the installation.

- 6) On the Start Copying Files dialog box, click **Next**.
At this time, DB2 Connect begins to install on your system.

When the installation is complete, the Setup Complete dialog box opens.

- 7) Click **Finish**.

The Steps dialog box opens.

- c. Click **Exit**.
- d. Reboot your system.

2. Configure your database connections

Note: This step is performed twice: once for the RunTime database and again for the CustTime database.

- a. Open the Client Configuration Assistant in the IBM DB2 program folder.
- b. If the Welcome dialog box opens, click **Add Database** . If the Welcome dialog box does not open, on the Client Configuration Assistant dialog box, click **Add**. The Client Configuration Assistant dialog box opens.
- c. On the Add Database Wizard dialog box, select **Manually configure a connection** on the Source tab page.
 - 1) Click **Next**.
The Protocol tab page opens.
 - 2) Select **TCP/IP**.
The TCP/IP tab page opens.
 - 3) Specify the name of the system or the IP address of the system that contains the databases, and specify the port number of the database.
This information must be obtained from your system administrator.
 - 4) On the Database tab, specify the name of the database.
The name is obtained from your system administrator. The default value for the RunTime database is EDIEC31E. The default value for the CustTime database is EDICT31E.
 - 5) Click **Finish**.
A Confirmation dialog box opens. Use this dialog to test your connection. If you are not able to connect successfully and can not resolve the issue, contact your system administrator.

3. Install WebSphere Data Interchange Client

Install WebSphere Data Interchange Client if it is not already installed.

4. Identify your system to WebSphere Data Interchange Client

- a. Start WebSphere Data Interchange Client.
- b. From the menu, click **View > EDI Systems**.
- c. Create a new EDI System that utilizes your databases.
 - 1) From the menu, click **File > New**, or click **Create New Document**.

An EDI System dialog box opens.

- a. Specify a name for the new system.
- b. Select the Data Source Name for the CustTime database.
- c. Specify the database qualifier for the CustTime database.

The qualifier is sometimes referred to as the schema. The default value for the CustTime database is CustTime. Confirm this value with your system administrator.

- d. Select the Data Source Name for the RunTime database.
- e. Specify the database qualifier for the RunTime database.

The qualifier is sometimes referred to as the schema. The default value for the RunTime database is EDIENU31. Confirm this value with your system administrator.

- f. Identify the platform that the WebSphere Data Interchange Server runs on.
- g. Select a color to be associated with this EDI System.

Associating unique colors with each EDI System helps the user to easily identify which system they are working with in a multi-system environment.

- h. Click **OK**.

The new system is saved.

- d. Exit and restart WebSphere Data Interchange Client.
- e. Click **Setup**.

- f. Enter the user ID and password for each of the CustTime and RunTime databases, if requested.

If the Setup Functional Area window opens, the connection to your databases is complete. If there is an error, ensure your EDI System is set up correctly. An incorrect database qualifier is a common problem.

Installation

Chapter 3. Running the WebSphere Data Interchange Server

The chapter provides instructions for running WebSphere Data Interchange Server.

Running from the command line

WebSphere Data Interchange Server always reads commands from STDIN and writes the results to STDOUT—these are treated as STREAMs. When invoked from the command line, the command line processor automatically opens STDIN and STDOUT, piping them wherever the user requests. You typically prepare a file of PERFORM commands for input and redirect the input from that file. You would probably redirect the STDOUT to a file.

An example of this usage is shown below:

```
ediservr < commands.txt > results.txt
```

Where `commands.txt` is the input file and has PERFORM commands, and `results.txt` contains the output from `ediservr.exe`.

The commands files consist of a set of WebSphere Data Interchange commands separated by semicolons. Every command is terminated with a semicolon.

The first command is always a SET, and the second command is always INIT. These are followed by a series of Set file commands that specify the input and output files for the following PERFORM command(s). The SET FILE PERFORM sequence can be repeated as many times as required.

A typical command file is as follows:

```
SET plan(EDIEC31E) userid(xxxxxx) password(xxxxx);  
INIT;  
SET FILE(APPFILE,c:\ne62\test\receive\input\appfile.txt);  
SET FILE(EDIFILE,c:\ne62\test\receive\input\edifile.txt);  
SET FILE(PRTFILE,c:\ne62\test\receive\output\prtfile.txt);  
SET FILE(EXPFILE,c:\ne62\test\receive\output\expfile.txt);  
SET FILE(TRKFILE,c:\ne62\test\receive\output\trkfile.txt);
```

Running the WebSphere Data Interchange Server

```
SET FILE(ENVEXCP,c:\ne62\test\receive\output\excpfile.txt);
SET FILE(RPTFILE,c:\ne62\test\receive\output\rptfile.txt);
SET FILE(FAKFILE,c:\ne62\test\receive\output\fakfile.txt);
SET FILE(QRYFILE,c:\ne62\test\receive\output\qryfile.txt);
PERFORM DEENVELOPE AND TRANSLATE WHERE
FILEID(EDIFILE) APPFILE(APPFILE) RAWDATA(Y);
TERM;
```

Description: commands.txt

SET command:

Sets up the environment

plan(EDIEC31E): To point to the WebSphere Data Interchange database

userid(XXXXX): User ID for the database

password(XXXX): Password for the user ID account

INIT command:

Loads the startup information and connect to the database using parameters defined above.

SET FILE (LogicalFileName, RealFileName):

Defines various INPUT and OUTPUT files needed for translation. The LogicalFileName of each file is assigned with a RealFileName that includes the complete path. Depending on the type of PERFORM command used, some files are mandatory, while others are optional.

PERFORM:

Issues standard perform commands. This command uses LogicalFileNames for various files used.

TERM:

Disconnects from the database and frees all allocated memory.

Description: results.txt (STDOUT)

If translation is successful, then contents of the file will be as follows:

DI Translator Started, build date: Feb 19 2002

DI Translator processed your request.

DI Translator shutdown

If translation fails, the contents of the file will be as follows:

DI Translator Started, build date: Feb 19 2002

Running the WebSphere Data Interchange Server

DI Translator Error. RC= "errorcode" , ERC="extended
return code"

DI Translator shutdown

Triggering from an MQSeries Queue

The WebSphere Data Interchange adapter program is installed as part of WebSphere Data Interchange for Multiplatforms Version 3.1. The configuration scripts provided set up the necessary queues and definition objects. The adapter uses MQSeries Triggering to know when messages need processing. When a message is put to an application queue, a trigger message is created. The MQSeries trigger monitor receives the message and executes the adapter. The adapter then passes the information needed to process the application message to the WebSphere Data Interchange server/translator. Application messages are committed, rolled back, or moved to the Queue Manager's Dead Letter queue, conditioned by the return codes from the WebSphere Data Interchange Server. The adapter will wait the user-configured time interval for any successive messages, and then terminate. The trigger monitor then restarts WebSphere Data Interchange adapter upon receipt of another trigger message.

Base MQSeries support architecture uses six MQSeries queues, three input and three output.

Input queues	Output queues
EDI_IN	EDI_OUT
ADF_IN	ADF_OUT
XML_IN	XML_OUT

The wdi.mqcommands file in the samples directory contains all MQ Service Command (MQSC) instructions for creating the needed queues.

Note: Necessary queues and definition objects are created in the default queue manager.

To create these six queues and configure the input queues for triggering, run the wdicommand script. The specific configuration scripts for integrating with other products, such as WebSphere MQSeries Integrator, are available at <http://www.ibm.com/websphere/datainterchange>.

Running the WebSphere Data Interchange Server

Each step of the trigger program is coordinated with a message exits dll called msgExits.dll that must reside somewhere in the binaries path at runtime.

The message exits dll can instruct the trigger program to skip messages, terminate, take or skip a syncpoint, and so on. It can be used to customize the behavior of the adapter to route failed messages to a special queue, or to notify someone if a failure in translation occurs. The interface to the message exits dll is documented in Adapter user exits on page 23.

When triggered, the adapter:

1. Reads the wdi.properties file for runtime directories.
2. Calls the trigger startup exit msgTrigger() if present and proceeds based on the return code from the exit.
3. Initializes WebSphere Data Interchange. If WebSphere Data Interchange cannot be initialized, the adapter turns triggering off for the queue and terminates.
4. Sets the name of the file that the message will be received into, which is *datadirectory*(from property files)/*rcvdirectory*(from property files)/*MQSeries message ID*(from MQMD).rcv.
5. For each message on the queue:
 - a. Browses the data queue to get the information on the next available message.
 - b. Calls the message tracking exit if present, and passes it the browse data. The message exit can return the batch ID to be used and an indicator of whether to proceed or to skip this message.
 - c. If OK to proceed, calls WebSphere Data Interchange with a PERFORM RECEIVE AND PROCESS ONEMESG(Y) WHERE REQID(mq_queue_name[1-16]) BATCHSET(batchid).
 - d. Upon returning from WebSphere Data Interchange, calls the msgTransform() exit with the return codes. If the return code from the exit instructs the trigger program not to proceed normally, then do what the return code is documented to mean in the adapter user exits on page 23, otherwise do the following:
 - If translation is acceptable (rc = 0), then execute a syncpoint.

Running the WebSphere Data Interchange Server

- If translation is not acceptable ($r < 0$), the adapter posts the message to the dead letter queue defined within MQSeries. Then execute a syncpoint.
- e. Moves on to the next message (restarts the process at step a).
6. When no more messages arrive within the specified interval (see Adapter user exits below), call the `msgTerminate` user exit (if one exists). If it indicates so, proceed with termination, then terminate WebSphere Data Interchange, and then the adapter itself.

Adapter user exits

To modify or monitor the behavior of the adapter, you can implement the adapter user exits. The WebSphere Data Interchange adapter loads the library, if found in the `bin` directory, and calls the exit functions. The shared library must be named `msgExits.dll` (`msgExits.so` on UNIX®) and should be compiled using the native compiler for the target platform (for example, Microsoft Visual C++ for Windows).

`msgExits.dll` interface:

- `bool msgTrigger(const char* pszTriggerMessage , void * pvExitContext);` - Called when the trigger program is started. Passes the trigger message TQTC2. Accepts a context that will be passed into all subsequent calls. The return value indicates whether to continue or terminate.
- `bool bSkip msgArrival(void* pvExitContext, char*pszSessionID)` - Message tracking exit that will be called just before attempting to get the next message. It can browse the queue for any information required and then pass back a session ID for WebSphere Data Interchange to use as the Batch ID. The return value indicates whether to process the message or skip it.
- `bool bProceed msgTransform(void* pvExitContext, long rc, long cbcrc, long ccberc)` - Results of the transformation.

Return values:

- `SYNC_CONTINUE` - syncpoint and then continue processing.
- `SYNC_TERM` - syncpoint and then terminate.
- `CONTINUE` - do not syncpoint, but continue processing.
- `TERMINATE` - terminate without taking a syncpoint.

Running the WebSphere Data Interchange Server

- `bool bOK msgTerminate(void* pvExitContext) - OK to terminate?`

Return values from `msgTerminate`:

- `#define SYNC_CONTINUE 0x0000`
- `#define SYNC_TERM 0x0001`
- `#define CONTINUE 0x0002`
- `#define TERMINATE 0x0003`

A configuration file is installed in the `wdi/bin` directory:

```
wdi.properties
```

The WebSphere Data Interchange Server runtime information is stored in a properties file located in the WDI bin directory. The installation default values for Windows are listed below. The default values for AIX are similar.

```
runtimedirectory=C:\Program Files\IBM\wdi\bin
datadirectory=C:\Program Files\IBM\wdi\RunTime
dtddirectory=DTD
prtdirectory=PRT
appdirectory=APP
edidirectory=EDI
xmldirectory=XML
aexdirectory=AEX
rptdirectory=RPT
fakdirectory=FAK
qrydirectory=QRY
xexdirectory=XEX
wrkdirectory=WRK
rcvdirectory=RCV
plan=EDIEC31E
*userid=user
*userpassword=password
Languagecode=ENU
waitinterval=10000
```

* Only required if you want to connect to the database using a different authorization ID than the one the adapter process is running under.

All files created and used during run time are created under the data directory. If you want WebSphere Data Interchange to

Running the WebSphere Data Interchange Server

create files in a different directory, then change the data directory value in `wdi.properties` to be the new directory you created. This directory can be another drive on Windows or file system on AIX.

The `waitinterval` value specifies the number of milliseconds the adapter waits for messages on the Application queue before terminating. Once the adapter terminates, another trigger message restarts the adapter.

Calling from a C++ program

This section provides an example of how to use the WebSphere Data Interchange C++ API. This example includes all the source code necessary to build a C++ program, using the WebSphere Data Interchange C++ API, to send several PERFORM commands to the WebSphere Data Interchange product.

Elements of the C++ API

The C++ API is made up of several classes that are all defined in the `diapi.h` header file shipped with the WebSphere Data Interchange product. The classes that make up the API are:

- `CSyncTranslator`
- `CASyncTranslator`
- `CRemoteTranslator`
- `CDIEnvironment`
- `CDIRequest`

Once a program includes the `diapi.h` header file, it can use these objects to interact with the WebSphere Data Interchange translator by passing in PERFORM commands to either a `CSyncTranslator`, `CASyncTranslator`, or `CRemoteTranslator` object. The three different types of translator objects that can be created are as follows:

- *CSyncTranslator*
The `CSyncTranslator` provides access to the translator in a synchronous manner. The process waits for each command to complete before allowing the next command to be performed.

Running the WebSphere Data Interchange Server

Methods:

CSyncTranslator(void) (Constructor)

Instantiates a new *CSyncTranslator* object. This method takes no arguments.

enum eResult Initialize(CDIEnvironment& env)

Causes the translator to be initialized. This method must be called before any transactions can be processed. This method takes a *CDIEnvironment* object that contains information about the system, such as database information (plan, user ID, password) and system information (language). This method returns an enumerated type that contains success or failure information about the method invocation.

enum eResult Terminate()

Terminates the translator and causes it to free any memory that was allocated during the translation process. This method takes no arguments and returns an enumerated type with information about the success or failure of the method invocation.

virtual enum eResult ProcessRequest(CDIRequest& req)

Initializes a PERFORM command to be processed by the translator. This method takes a *CDIRequest* object that has been initialized with a perform command. The enumerated type returned by the function can be used to determine the success or failure of the PERFORM command.

virtual long GetRetCode(void)

Gets the return code from the last translator action performed. This method takes no arguments.

virtual long GetExtRetCode(void)

Accesses the extended return code from the last translator action performed. This method takes no arguments.

- *CASyncTranslator*

The *CASyncTranslator* provides asynchronous access to the translator. This method allows your program to begin processing on several transactions at once without waiting for the previous PERFORM command to complete.

Running the WebSphere Data Interchange Server

Methods:

CAsyncTranslator(short sMaxReqs=10, short nNice=0)

Takes two arguments: *sMaxReqs* and *nNice*. *sMaxReqs* specifies the maximum number of requests that can be made. *nNice* specifies a nice value any process created by the translator during this session.

enum eResult Initialize(CDIEnvironment& env)

Initializes the translator using the *CDIEnvironment* argument. The *CDIEnvironment* object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

enum eResult Terminate()

Initializes the translator using the *CDIEnvironment* argument. The *CDIEnvironment* object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

enum eResult ProcessRequest(CDIRequest& req.)

Passes the perform command contained in the *CDIRequest* object to the translator to be executed.

short GetMaxRequests()

Returns the maximum number of requests that can be executed at one time. This value limits the number of translators that can be started by this object.

short GetCurrentRequests()

Returns the number of requests being processed.

enum eResult UpdateCurReqCnt(void)

Causes the current request count to be updated.

- *CRemoteTranslator*

The *CRemoteTranslator* provides access to a WebSphere Data Interchange translator running on a remote system. The *CRemoteTranslator* is like the *CAsyncTranslator*, except its constructor takes the hostname of the remote system as an additional argument to its constructor.

Running the WebSphere Data Interchange Server

Methods:

CRemoteTranslator(char pszHost,short sMaxReqs=10)*

Creates a new instance of the CRemoteTranslator class that can communicate with a remote server using TCP/IP sockets.

enum eResult Initialize(CDIEnvironment& env)

Initializes the translator using the CDIEnvironment argument. The CDIEnvironment object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

enum eResult Terminate()

Initializes the translator using the CDIEnvironment argument. The CDIEnvironment object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

enum eResult ProcessRequest(CDIRequest& req.)

Passes the PERFORM command contained in the CDIRequest object to the translator to be executed.

short GetMaxRequests()

Returns the maximum number of requests that can be executed at one time. This value limits the number of translators that can be started by this object.

short GetCurrentRequests()

Returns the number of requests being processed.

enum eResult UpdateCurReqCnt(void)

Causes the current request count to be updated.

- *CDIEnvironment*

The CDIEnvironment class encapsulates all the system settings needed by the CSyncTranslator, CAsyncTranslator, and CRemoteTranslator during their initialization. The CDIEnvironment class must be instantiated and then passed to the initialize method of one of the translator objects.

Running the WebSphere Data Interchange Server

Methods:

void SetSys(char pszVal)*

Identifies the installation-defined DataInterchange for MVS systems used to run the EDIUTILV utility. The default is DIENU.

void SetAppl(char pszVal)*

Identifies the Application ID to run the DataInterchange utility. This keyword also identifies the logfile specified by the ACTLOGS profile. If you specify this parameter, the activity log profile must contain a matching entry to define which log file is used for recording errors and events pertaining to the application. The two APPLID values shipped with DataInterchange are:

- EDIFFS (default)
 - Associated with the LOGFFS ddname
 - The default APPLID and log when using the utilities
- EDIMP
 - Associated with the LOGEDI ddname
 - The APPLID and log used during online DataInterchange processing

void SetLang(char pszVal)*

Identifies the language profile to use as specified in the LANGPROF profile. The value you specify with the SetLang method must match one of the values in the LANGPROF profile. The LANGPROF that ships with WebSphere Data Interchange is ENU.

void SetPlan(char pszVal)*

Identifies the DB2 plan that WebSphere Data Interchange is to use to access its database tables.

void SetEdiDataQueueName(char pszVal)*

Identifies the routing queue for completion message from the translator. When the CAsyncTranslator completes the processing of its EDI data it will send a completion message to this queue.

void SetAppDataQueueName(char pszVal)*

Identifies the routing queue for completion message for ADF data coming from the translator. When the

Running the WebSphere Data Interchange Server

CAsyncTranslator completes the processing of an ADF, it will send a completion message to this queue.

void SetEdiErrorQueueName(char pszVal)*

Identifies a queue where errors identified during the processing of EDI data can be sent. When the CAsyncTranslator encounters errors, an EDI data message will be sent to this queue.

void SetAppErrorQueueName(char pszVal)*

Identifies a queue where errors identified during the processing of ADF data can be sent. When the CAsyncTranslator encounters errors, an ADF data message will be sent to this queue.

void SetHostName(char pszVal)*

Sets the hostname of a remote WebSphere Data Interchange translator. The value set in this method is only used with the CRemoteTranslator.

void SetHostPort(int nVal)

Identifies the port number used by a remote WebSphere Data Interchange translator for network communication. The value set by this method is only used by the CRemoteTranslator class.

void SetUser(char pszVal)*

Sets the database user ID needed to access the DB2 database. This only needs to be set if the user ID of the person running the program does not have the necessary authority to access the database.

void SetPassword(char pszVal)*

Sets the password to be used by the translator to access the WebSphere Data Interchange database. This is only required if the user ID of the person running the program does not have the authority to access the database.

void SetRouterType(enum CDIMsgQueue::qtype enVal)

Sets the type of routers (queue) for the completion messages. This method can accept the following values:

file //File
pipe //Named pipe
socket //TCP/IP socket
email //Email address

Running the WebSphere Data Interchange Server

void SetUnitOfWork(enum eUnitOfWork enVal)

Defines a unit of work to the translator. This method allows the application programmer to define the point at which Commits should be done. Possible values for this function are:

- | | |
|--------------|--|
| eTransaction | Commits should be done after every transaction |
| eEnvelope | Commits should be done after every envelope is encountered |
| eNoCommit | No commits are performed |

void SetInterfaceType(enum eInterfaceType enVal)

Possible values for the function are:

- | | |
|------------|---|
| eITCmdLine | //Invoked by command line |
| eITApi | //Invoked by API |
| eITWeb | //Invoked by Web server (not supported) |

- *CDIRequest*

The *CDIRequest* represents a request for translation that can be submitted to a translator to be processed. The *CDIRequest* object names the files necessary to perform a translation as well as the perform statement to be executed. This is a list of the files that can be associated with a *CDIRequest* object:

- Application File
- EDI File
- Tracking File
- Exception File
- EDI Except File
- Print File
- Report File
- Query File
- Work File
- Functional Acknowledgment File

Methods:

CDIRequest(void)

The constructor for *CDIRequest* builds an instance of the *CDIRequest* object. This method does not take any arguments.

Running the WebSphere Data Interchange Server

void ClearOutput(void)

Clears all the output fields of the request object.

void SetAppFile(char pszFile)*

Sets the name of the application file for this request.

void SetEdiFile(char pszFile)*

Sets the name of the EDI file for this request.

void SetTrackingFile(char pszFile)*

Sets the name of the tracking file for this request.

void SetExceptionFile(char pszFile)*

Sets the name of the exception file where ADF data can be written if a fatal error is encountered during the processing of the PERFORM command.

void SetEdiExceptFile(char pszFile)*

Sets the name of the exception file where EDI data can be written if a fatal error is encountered during the processing of the PERFORM command.

void SetPrintFile(char pszFile)*

Sets the name of the print file where status information about a completed translation can be written.

void SetReportFile(char pszFile)*

Sets the name of the report file where reports and printouts that you have requested can be stored.

void SetQueryFile(char pszFile);*

Sets the name of the file where results from a QUERY or DATA EXTRACT can be stored.

void SetWorkFile(char pszFile)*

Sets the name of a file that can be used as a temporary workspace during the translation.

void SetFunAckFile(char pszFile)*

Sets the name of the file that you want to use for returning functional acknowledgments for the deenveloped transactions.

API return codes

Refer to Appendix A, "C++ and Java API return codes" on page 51 for a list of the return codes.

WebSphere Data Interchange API example

This section outlines an example program that uses the C++ API for WebSphere Data Interchange. This example is made up of one source file that initializes two CDIRequest objects with two PERFORM statements. The sample script starting on page 34 contains a listing of the main items for this example.

This program begins by creating CSyncTranslator, CDIEnvironment, and two CDIRequest objects (EDI2ADFRequest and ADF2EDIRRequest) to handle the two different requests. Next, the two requests were initialized with the filenames needed to process the requests, and the setPerformCommand method was used to set the desired PERFORM commands to execute.

Once the PERFORM commands were created and the translator was initialized, the ProcessRequest method was called to execute the PERFORM commands. When that was done, the return codes were checked and the translator was terminated and the program was exited.

Building the Example

This example can be built on the different platforms supported by WebSphere Data Interchange.

AIX

1. Copy the files apiexamp.cpp and apiexamp.mk from **/usr/wdi/DIv3.1/samples** to your home directory or any other work directory where you have permissions to write files.
2. Issue the command `make -f apiexamp.mk`.
This will build an executable named apiexamp.
3. Import the demo maps from demo850clrecvsuw.eif and demo850clsendsuw.eif using the WebSphere Data Interchange Client.

Running the WebSphere Data Interchange Server

Windows

On the Microsoft Windows platform, you have several choices of compilers. In this example, the Microsoft VisualC++ compiler is used to build the API example:

1. Create an empty win32 console project within Microsoft Visual C++.
2. Add apiexamp.cpp to the project.
3. Build the project you created.
4. Execute the newly built apiexamp executable.

Sample script

```
#include <iostream.h>
#include "diapi.h"
/*-----*/
/* Main program                                     */
/*-----*/
void pause(void);

int main ()
{
    CDIEEnvironment    aCDIEEnvironment;
    CDIRequest         anADF2EDIRequest;
    CDIRequest         anEDI2ADFRequest;
    CSyncTranslator    aCSyncTranslator;
    enum eResult       rc;

    //Define the Data Interchange Environment
    aCDIEEnvironment.SetPlan("EDIEC31E");
    aCDIEEnvironment.SetLang("ENU      ");

    // Initialize the translator:
    rc = aCSyncTranslator.Initialize(aCDIEEnvironment);
    cout << endl << endl << "TestInitialize:  rc=" << rc
         << ", zccbrc=" << aCSyncTranslator.GetRetCode()
         << ", zccberc=" << aCSyncTranslator.GetExtRetCode()
         << endl;

    pause();

    // Name the input and output files for an EDI to ADF request
    anEDI2ADFRequest.SetAppFile("demo850.app");
    anEDI2ADFRequest.SetEdiFile("demo850.edi");
    anEDI2ADFRequest.SetTrackingFile("demo850.trk");
    anEDI2ADFRequest.SetExceptionFile("demo850.aex");
```

Running the WebSphere Data Interchange Server

```
anEDI2ADFRequest.SetPrintFile("demo850.prt");
anEDI2ADFRequest.SetFunAckFile("denvtr.fak");
anEDI2ADFRequest.SetWorkFile("denvtr.wrk");
anEDI2ADFRequest.SetReportFile("denvtr.rpt");
anEDI2ADFRequest.SetQueryFile("denvtr.qry");

// Name the input and output files for an ADF to EDI request
anADF2EDIRequest.SetAppFile("denvtr.app");
anADF2EDIRequest.SetEdiFile("denvtr.edi");
anADF2EDIRequest.SetTrackingFile("denvtr.edi");
anADF2EDIRequest.SetExceptionFile("denvtr.aex");
anADF2EDIRequest.SetPrintFile("denvtr.prt");
anADF2EDIRequest.SetFunAckFile("denvtr.fak");
anADF2EDIRequest.SetWorkFile("denvtr.wrk");
anADF2EDIRequest.SetReportFile("denvtr.rpt");

// Set the perform commands to be executed:
//                                     ADF-TO-EDI      TEST      CASE:
*****
anADF2EDIRequest.SetPerformCmd
("PERFORM TRANSLATE AND ENVELOPE WHERE APPFILE(APPFILE) "
 "FILEID(EDIFILE) RAWTEST(Y) RAWFMTID(MMTHL7) PURGINT(-1)");
//                                     EDI-TO-ADF      TEST      CASES:
*****
anEDI2ADFRequest.SetPerformCmd
("PERFORM DEENVELOPE AND TRANSLATE WHERE FILEID(EDIFILE) "
 "APPFILE(APPFILE) RAWDATA(Y) PURGINT(-1)");

// Ask the synchronous translator to process the EDI to ADF Request:
cout << "EDI to ADF translation" << endl;
rc = aCSyncTranslator.ProcessRequest(anEDI2ADFRequest);

// Confirm the input and output names and print the return codes:
cout << endl << endl << "App File      : "
  << anEDI2ADFRequest.GetAppFile() << ", "
  << anEDI2ADFRequest.GetAppFileLen() << endl;
cout << "EDI File      : " << anEDI2ADFRequest.GetEdiFile() << ", "
  << anEDI2ADFRequest.GetEdiFileLen() << endl;
cout << "Report File   : "
<< anEDI2ADFRequest.GetReportFile() << ", "
  << anEDI2ADFRequest.GetReportFileLen() << endl;
cout << "Exception File : "
<< anEDI2ADFRequest.GetExceptionFile() << ", "
  << anEDI2ADFRequest.GetExceptionFileLen() << endl;
cout << "Print File    : "
  << anEDI2ADFRequest.GetPrintFile() << ", "
  << anEDI2ADFRequest.GetPrintFileLen() << endl;
```

Running the WebSphere Data Interchange Server

```
cout << "Tracking File : "  
    << anEDI2ADFRequest.GetTrackingFile() << ", "  
    << anEDI2ADFRequest.GetTrackingFileLen() << endl;  
cout << "Query File : "  
    << anEDI2ADFRequest.GetQueryFile() << ", "  
    << anEDI2ADFRequest.GetQueryFileLen() << endl;  
cout << "Work File : " << anEDI2ADFRequest.GetWorkFile() << ", "  
    << anEDI2ADFRequest.GetWorkFileLen() << endl;  
cout << "FunAck File : "  
    << anEDI2ADFRequest.GetFunAckFile() << ", "  
    << anEDI2ADFRequest.GetFunAckFileLen() << endl;  
cout << endl << endl << "ProcessRequest: rc=" << rc  
    << ", zccbrc=" << aCSyncTranslator.GetRetCode()  
    << ", zccberc=" << aCSyncTranslator.GetExtRetCode();  
  
cout << "EDI to ADF request complete" << endl;  
pause();  
  
    // Ask the synchronous translator to process the ADF to EDI  
Request:  
    cout << "ADF to EDI translation" << endl;  
    rc = aCSyncTranslator.ProcessRequest(anEDI2ADFRequest);  
  
    // Confirm the input and output names and print the return codes:  
cout << endl << endl << "App File : "  
    << anADF2EDIRequest.GetAppFile() << ", "  
    << anADF2EDIRequest.GetAppFileLen() << endl;  
cout << "EDI File : "  
    << anADF2EDIRequest.GetEdiFile() << ", "  
    << anADF2EDIRequest.GetEdiFileLen() << endl;  
cout << "Report File : "  
    << anADF2EDIRequest.GetReportFile() << ", "  
    << anADF2EDIRequest.GetReportFileLen() << endl;  
cout << "Exception File : "  
    << anADF2EDIRequest.GetExceptionFile() << ", "  
    << anADF2EDIRequest.GetExceptionFileLen() << endl;  
cout << "Print File : "  
    << anADF2EDIRequest.GetPrintFile() << ", "  
    << anADF2EDIRequest.GetPrintFileLen() << endl;  
cout << "Tracking File : "  
    << anADF2EDIRequest.GetTrackingFile() << ", "  
    << anADF2EDIRequest.GetTrackingFileLen() << endl;  
cout << "Query File : "  
    << anADF2EDIRequest.GetQueryFile() << ", "  
    << anADF2EDIRequest.GetQueryFileLen() << endl;  
cout << "Work File : "  
    << anADF2EDIRequest.GetWorkFile() << ", "  
    << anADF2EDIRequest.GetWorkFileLen() << endl;
```

Running the WebSphere Data Interchange Server

```
cout << "FunAck File      : "  
      << anADF2EDIRequest.GetFunAckFile() << ", "  
      << anADF2EDIRequest.GetFunAckFileLen() << endl;  
cout << endl << endl << "ProcessRequest: rc=" << rc  
      << ", zccbrc=" << aCSyncTranslator.GetRetCode()  
      << ", zccberc=" << aCSyncTranslator.GetExtRetCode()  
      << endl;  
  
// Terminate the translator:  
cout << "Now terminating the translator to free up any resources"  
      << endl;  
rc = aCSyncTranslator.Terminate();  
cout << endl << endl << "Terminate: rc=" << rc  
      << ", zccbrc=" << aCSyncTranslator.GetRetCode()  
      << ", zccberc=" << aCSyncTranslator.GetExtRetCode();  
  
// Terminate and go home:  
cout << endl << endl;  
return(0);  
}  
  
void pause()  
{  
    char achar;  
    cout << "Hit enter to continue" << endl;  
    cin.get(achar);  
}
```

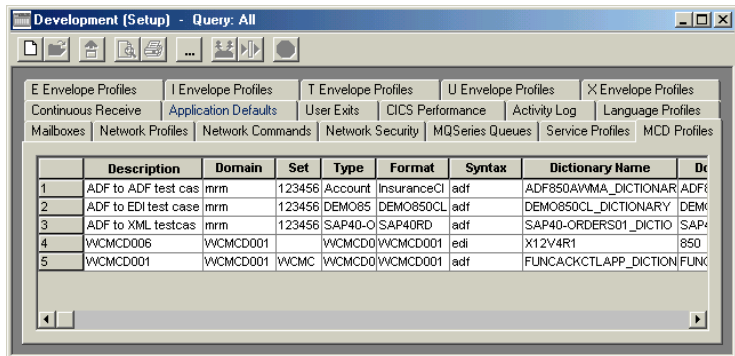
Running the WebSphere Data Interchange Server

Chapter 4. Message Content Descriptor profiles

With the Message Content Descriptor (MCD) profile, you can map the names of message definitions in other products, such as WebSphere Message Repository Manager (MRM) and Crossworlds, to the names of message definitions within WebSphere Data Interchange when communicating using the MQSeries Rule and Formatting Header 2 (MQRFH2). The MCD profile is also used when communicating with Java Message Service (JMS) clients. For more information about how the fields in the MCD map to the JMS API see “Mapping the MQRFH2 header to the JMS API” on page 57.

Setup overview

To set up and maintain MCD profiles through the MCD Profiles List window, click **Setup** on the WebSphere Data Interchange Client Navigator bar. The Setup window, which contains tabs for WebSphere Data Interchange Client’s profiles, opens. Click the **MCD Profiles** tab to open the MCD Profiles List window.



The screenshot shows a window titled "Development (Setup) - Query: All". It has a menu bar with icons and a toolbar. Below the toolbar is a tabbed interface with several tabs: "E Envelope Profiles", "I Envelope Profiles", "T Envelope Profiles", "U Envelope Profiles", "X Envelope Profiles", "Continuous Receive", "Application Defaults", "User Exits", "CICS Performance", "Activity Log", "Language Profiles", "Mailboxes", "Network Profiles", "Network Commands", "Network Security", "MQSeries Queues", "Service Profiles", and "MCD Profiles". The "MCD Profiles" tab is selected, displaying a table with the following data:

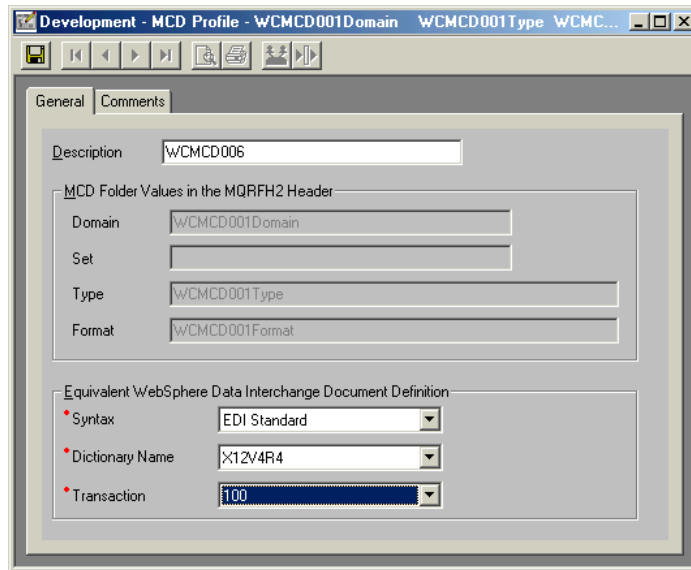
	Description	Domain	Set	Type	Format	Syntax	Dictionary Name	Dt
1	ADF to ADF test cas	mrm	123456	Account	InsuranceCl	adf	ADF850AWMA_DICTIONAR	ADF
2	ADF to EDI test case	mrm	123456	DEMO85	DEMO850CL	adf	DEMO850CL_DICTIONARY	DEMO
3	ADF to XML testcas	mrm	123456	SAP40-O	SAP40RD	adf	SAP40-ORDERS01_DICTIO	SAP
4	WCMCD006	WCMCD001		WCMCD0	WCMCD001	edi	X12V4R1	850
5	WCMCD001	WCMCD001	WCMC	WCMCD0	WCMCD001	adf	FUNCTIONCTLAPP_DICTIO	FUNC

This window displays a list of existing MCD profiles. Each row contains information about an MCD profile and each column contains data stored in the profile. Information in the columns displays in fields, drop-down lists, and check boxes in the MCD Profiles Editor window. The profile list window, however, also contains the date, time, and user ID of the last update.

Message Content Descriptor profiles

To display additional columns, click the scroll bar on the bottom of the screen to scroll to the right or left. To alter the columns that display on the screen, or to change which query is executed to produce the list, click **Modify Window Properties**. To create new queries, refer to the *DataInterchange Client User's Guide*.

To view a profile or to add or change the information in these fields, double-click the row of the profile you wish to work with. The MCD Profiles Editor window opens, with the General tab in front.



The MCD Profiles Editor window contains two tabs: General and Comments. Use the General tab to add or change information contained in the MCD profile. Use the Comments tab to add any comments you wish about the selected MCD profile.

Following are detailed procedures for creating new MCD profiles. For information on viewing, editing, and deleting profiles, see the *DataInterchange Client User's Guide*.

Creating MCD profiles

Create a new MCD profile when exchanging documents using MQSeries message queues or JMS, and the message containing the document is prefixed with an MQRFH2 header. The MQRFH2 header contains an MCD folder. The name of a WebSphere Data Interchange document definition occurs in an MCD profile, along with the corresponding values from the MCD of the MQRFH2 header.

1. On the WebSphere Data Interchange Client Navigator bar, click **Setup**.

The Setup window opens.


2. Click the **MCD Profiles** tab.

A list of the existing MCD profiles displays.

3. On the tool bar, click **New**.

The MCD Profiles Editor window opens with the **General** tab in front and all fields blank.

4. Complete the fields on the **General** tab. Required fields are preceded by a red dot.

Click  for field descriptions.

5. Click the **Comments** tab and type any comments you have about the MCD profile into the **Comments** field.
6. From the File menu, click **Save** to save the profile.

WebSphere Data Interchange saves the new MCD profile to the database.

7. Close the editor.

Message Content Descriptor profiles

Chapter 5. Service profiles

The purpose of the Service profile is to allow the customer to enter a utility command and all the files that will be used during execution of that command. There are specific fields for fixed names, such as the print file (PRTFILE), and short name/long name pairs for times when both the short and long names are user defined, such as input and output files.

Though this feature is designed to work in a general manner and has many possible applications, it was primarily designed for situations where an input source contains both documents that require single translation and documents that require double translation. With command chaining, you can set up a double translation by routing the output from the first translation to a file with the same name as a Service profile.

The name you give the Service profile is its logical name. If another command writes information to the file associated with this logical name, the PERFORM command is executed after that command completes, chaining the commands together.

Substitution keywords

Several keywords that can be used in PERFORM commands are available to assist in creating command chains.

&MSD

&SET

&TYPE

&FORMAT - Comes straight from the MCD profile. They match the header in the MQ Integrator RFH2 header. If there is no RFH2 header on the message, or it does not match to something within the MCD profile, the **&FORMAT** is the Format field in the MQSeries MQMD header.

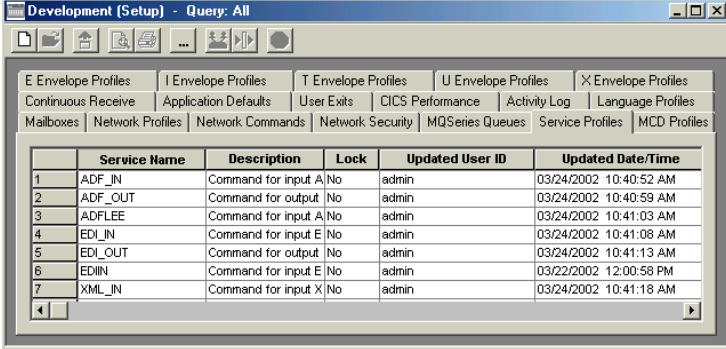
&SYNTAX

&DICTIONARY

&DOCUMENT - Corresponds to the WebSphere Data Interchange elements within the MCD profile. **&DOCUMENT** is the RAWFMTID in the case of ADF syntax.

Setup overview

To set up and maintain Service profiles through the Service Profiles List window, click **Setup** on the WebSphere Data Interchange Client Navigator bar. The Setup window, which contains tabs for WebSphere Data Interchange Client's setup profiles, opens. Click the **Service Profiles** tab, and the Service Profiles List window opens.

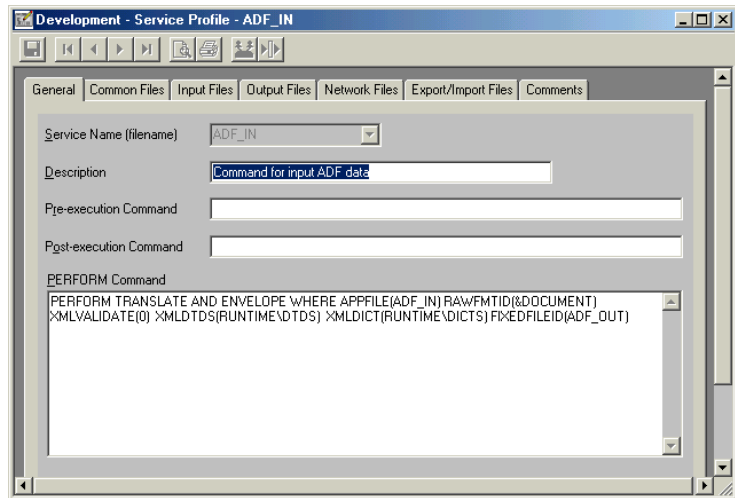


	Service Name	Description	Lock	Updated User ID	Updated Date/Time
1	ADF_IN	Command for input A	No	admin	03/24/2002 10:40:52 AM
2	ADF_OUT	Command for output	No	admin	03/24/2002 10:40:59 AM
3	ADFLEE	Command for input A	No	admin	03/24/2002 10:41:03 AM
4	EDI_IN	Command for input E	No	admin	03/24/2002 10:41:08 AM
5	EDI_OUT	Command for output	No	admin	03/24/2002 10:41:13 AM
6	EDIIN	Command for input E	No	admin	03/22/2002 12:00:58 PM
7	XML_IN	Command for input X	No	admin	03/24/2002 10:41:18 AM

This window displays a list of Service profiles. Each row contains information about a Service profile and each column contains data stored in the profile. Information in the columns displays in fields, drop-down lists, and check boxes in the Service Profiles Editor window. The profile list window, however, also contains the date, time, and user ID of the last update.

To display additional columns, click the scroll bar on the bottom of the screen to scroll to the right or left. To alter the columns that display on the screen, or to change which query is executed to produce the list, click **Modify Window Properties**. To create new queries, refer to the *DataInterchange Client User's Guide*.

To view a profile or to add or change the information in these fields, double-click the row of the profile you wish to work with. The Service Profiles Editor window opens, with the General tab in front.



The Service Profiles Editor window contains seven tabs: General, Common Files, Input Files, Output Files, Network Files, Export/Import Files, and Comments. Use the General tab to enter and change information contained in the Service profile. Use the other tabs to add other pertinent information about the Service profile. Whether you are running on a Windows or AIX server, be sure to follow the rules for names and path names for that server. You can use the relative or the absolute path name. The relative path name is to the current directory that the task was started from, or where the trigger started the task.

Following are detailed procedures for creating new Service profiles. For information on viewing, copying, editing, renaming, deleting, and printing profiles, see the *DataInterchange Client User's Guide*.

Creating Service profiles

Create a new Service profile when you need to set up a WebSphere Data Interchange PERFORM command to be invoked when an output file is closed at the end of other WebSphere Data Interchange command processing.

Within WebSphere Data Interchange, files have logical and physical names. At runtime, the logical names are mapped to the physical names.

Service profiles

Plan the chain of command, and then create the Service profile for each of the steps in your chain.

1. On the WebSphere Data Interchange Client Navigator bar, click **Setup**.

The Setup window opens.


2. Click the **Service Profiles** tab.

A list of the existing Service profiles displays.

3. On the tool bar, click **New**.

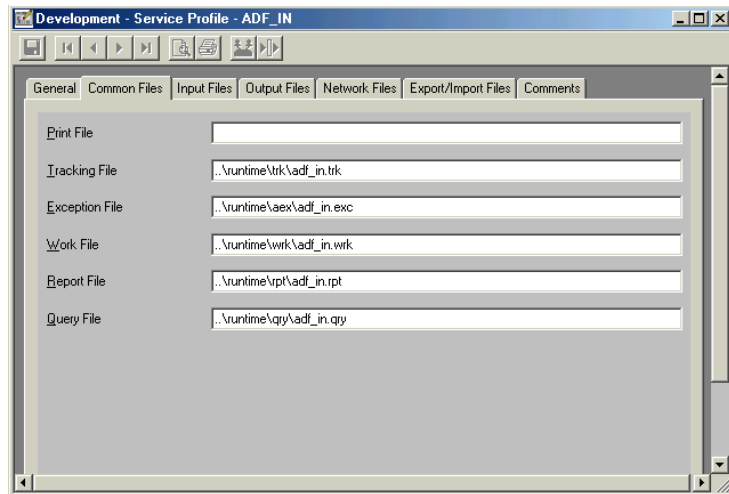
The Service Profiles Editor window opens with the General tab in front and all fields blank.

4. Complete the fields on the General tab. Required fields are preceded by a red dot.

Click  for field descriptions.

5. Click the **Common Files** tab.

The fields on the Common Files tab display.



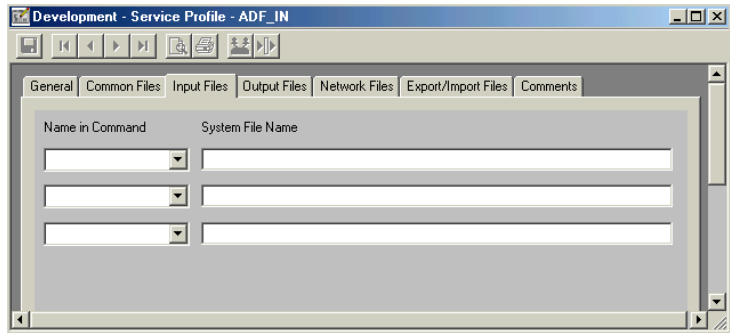
Note: If a file is not specified here, then the file from the previous command is used.

Tip: It is usually best not to specify the print file, so that all the messages for all the commands in the chain will go to the same file.

Service profiles


- Fill in the fields as needed. The Common Files tab contains only fixed names.
- Click the **Input Files** tab.

The fields on the Input Files tab display.



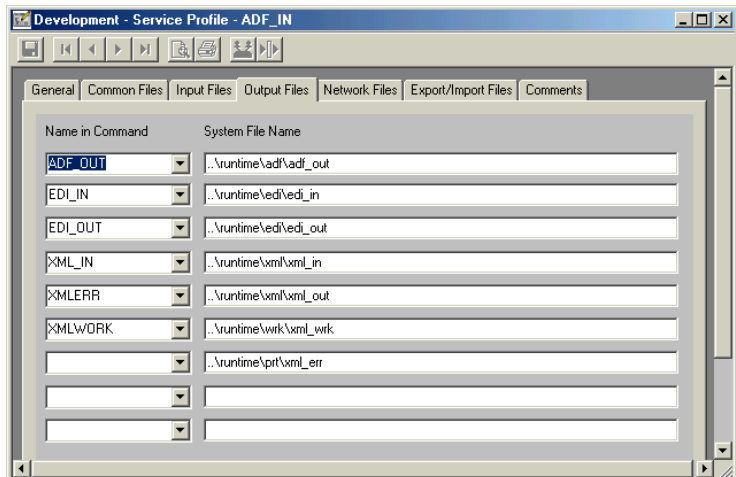
Note: This tab is only used if the input to the command is different than the file that triggered the command. The input file is usually the name of the file that invoked this command. That file is automatically available.

- Fill in the fields as needed. The Input Files tab contains logical and physical names.

Click  for field descriptions.

- Click the **Output Files** tab.


The fields on the Output Files tab display.



Service profiles

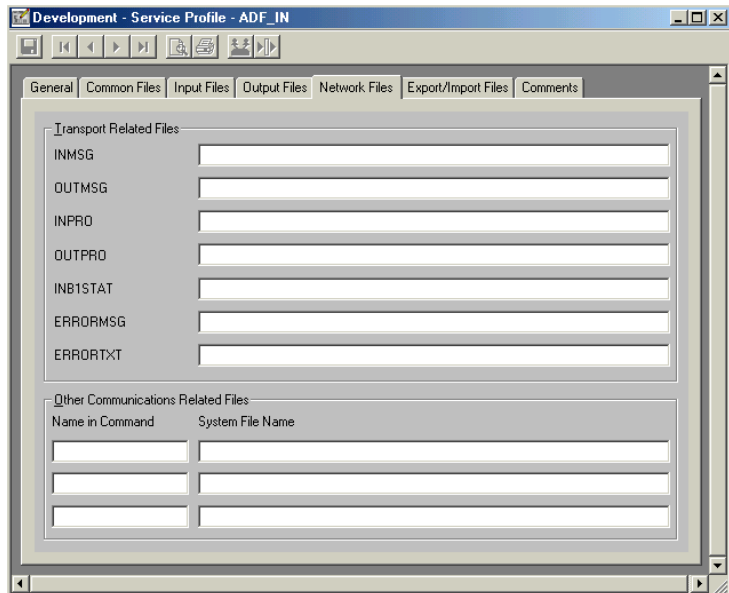
Note: You must specify all possible output files for the command.

10. Fill in the fields as needed. The Output Files tab contains logical and physical names.


Click  for field descriptions.

11. Click the **Network Files** tab.

The fields on the Network Files tab display.

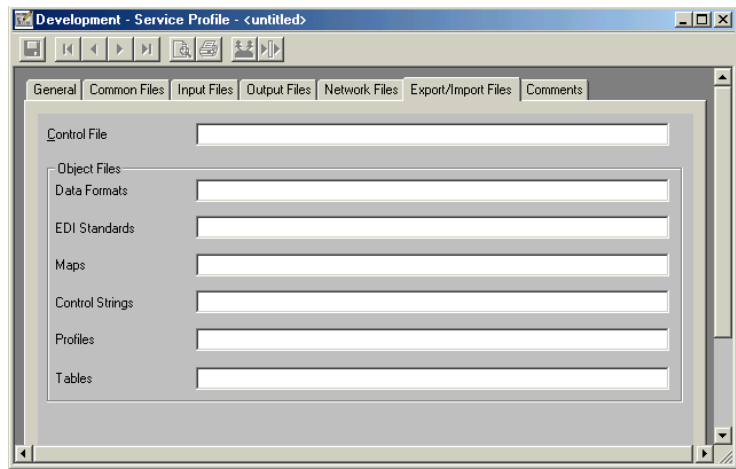


12. Fill in the fields as needed. The Network Files tab contains fixed names.

Click  for field descriptions.


13. Click the **Export/Import Files** tab.

The fields on the Export/Input Files tab display.



Note: You can specify all the object files to be the same file.

14. Fill in the fields as needed. The Export/Import Files tab contains fixed names.

Click  for field descriptions.

15. Click the **Comments** tab and add any comments you have about the Service profile into the Comments field.
16. From the file menu, click **Save** to save the profile.

WebSphere Data Interchange saves the new MCD profile to the database.

17. Close the editor.

Service profiles

Appendix A. C++ and Java API return codes

When a message is processed, return codes are generated by WebSphere Data Interchange. Return codes are written to an event log that can be queried using a PRINT EVENT LOG command. You can review a single transaction by specifying the Transaction ID or review a batch of transactions by specifying the Batch ID. You can also query the Transaction Store for specific transactions by using a PRINT TRANSACTION DETAILS command and specifying the Transaction ID.

Although all successful processing codes are written to the event log, only exception codes for serious errors are written to the event log. Sufficient information is provided to allow you to find further information in the DataInterchange Version 3.1 publications.

The return codes generated as follows:

Code	Definition
0-1	Processing was successful.
5	A terminal error occurred and the Solution Manager was shut down. Check the error log to see which error occurred. For more information on error codes, see Appendix B or refer to the <i>DataInterchange Programmer's Reference</i> .
6 or higher	A terminal error occurred and the Solution Manager was shut down. Check the error log to see which error occurred. For more information on error codes, see Appendix B or refer to the <i>DataInterchange Programmer's Reference</i> .

The exception codes consist of three separate codes:

- The API Return Code (documented below) usually indicates a programming error
- The Utility Return Code
- The Utility Extended Return Code (documented in *DataInterchange Messages and Codes*)

C++ and Java API return codes

Code	Definition
0	Good return.
1	Not used.
2	The request has been queued to a translation server.
3	Not used.
4	There was a problem with the service director. Check the event log. Then call GetRetCode() and GetExtRetCode() to find more information.
5	There was a problem executing the request. Check the event log. Then call GetRetCode() and GetExtRetCode() to find more information.
6	CDIMsgQueue::Open - Invalid router queue type. Valid types are file and pipe.
7	Check errno for more information.
8	CDIMsgQueue::Open - Queue name is NULL.
9	CDIMsgQueue::Open - Queue already open.
10	CDIMsgQueue::Open - Open on queue failed. Check errno for more information.
11	CDIMsgQueue::Write - The file descriptor is invalid.
12	CDIMsgQueue::Write - Attempted write to router queue failed. Check errno for more information.
13	CDIEnvironment::Parse - argc/argv parameters are invalid.
14	CDIEnvironment::Parse - Command line parameters to the ediservr program are invalid.
15	CSyncTranslator::Initialize - Translator already initialized. Initialize only once.

C++ and Java API return codes

Code	Definition
16	CSyncTranslator::Initialize - An attempt was made to initialize the translator without specifying a DB2 plan. Set the plan before initializing.
17	CSyncTranslator::Initialize - Failed to initialize on call to service director. Check the return code and extended return code by calling the GetRetCode() and GetExtRetCode() methods.
18	CSyncTranslator::ProcessRequest - A call was made to process a request, but the translator has not been initialized yet.
19	CSyncTranslator::Initialize - A call was made to process a request, but there is no request to process.
20	CSyncTranslator::Initialize - The unit of work specified for this request is invalid.
21	CSyncTranslator::SetFileName - The mandatory parameter logical filename is NULL.
22	CSyncTranslator::SetFileName - The mandatory parameter logical filename is an empty string.
23	CSyncTranslator::SetFileName - The mandatory parameter logical filename is >8 characters long.
24	CSyncTranslator::SetFileName - Error calling the service director to set the fully qualified name of one of the files used in a request. One of the directories specified is probably invalid.
25	CSyncTranslator::GetFileName - The mandatory parameter logical filename is NULL.
26	CSyncTranslator::GetFileName - The mandatory parameter logical filename is an empty string.

C++ and Java API return codes

Code	Definition
27	CSyncTranslator::GetFileName - The mandatory parameter logical filename is >8 characters long.
28	CSyncTranslator::GetFileName - The service director was not able to get the fully qualified name of one of the files used in a request.
29	CSyncTranslator::GetFileName - Error calling the service director to get the fully qualified name of one of the files used in a request.
30	CSyncTranslator::Terminate - A call was made to terminate the translator, but the translator has not been initialized yet.
31	CAsyncTranslator::Initialize - An attempt was made to initialize an asynchronous translator in a WIN32 environment. This is not supported at this time. Use CRemoteTranslator instead.
32	CAsyncTranslator::Initialize - An attempt was made to initialize an asynchronous translator that has already been initialized.
33	CAsyncTranslator::Initialize - The asynchronous translation server is already running.
34	CAsyncTranslator::Initialize - Error creating the command pipes to communicate with the asynchronous translation server. Check errno for more information.
35	CAsyncTranslator::Initialize - Error attempting to fork() to create the asynchronous translation server. Check errno for more information.
36	CAsyncTranslator::Initialize - Failed to load the program ediservr. Make sure it is in the PATH for executables. Check errno for more information in the child process only.

C++ and Java API return codes

Code	Definition
37	CAsyncTranslator::ProcessRequest - The translator has not been initialized yet and a call was made to ProcessRequest(). Before processing any requests, the translator must be initialized.
38	CAsyncTranslator::ProcessRequest - A call was made to process a request, but there is no request to process. Check the request type and PERFORM command.
39	CAsyncTranslator::ProcessRequest - The request has NOT been queued. The maximum number of requests have already been queued to this translation server. Select (or start) another translation server, wait for this server to finish its current request, or increase the max_requests parameter to allow more requests to be queued.
40	CAsyncTranslator::Write - An error occurred attempting to send a command to the translation server. Verify that the translation server is running. Check errno for more information.
41	CAsyncTranslator::UpdateCurReqCnt - An error occurred attempting to determine if there is any data in the response queue. Verify that the translation server is running. Check errno for more information.
42	CAsyncTranslator::UpdateCurReqCnt - An error occurred attempting to determine if there is any data in the response queue. Select returned an invalid code.
43	CAsyncTranslator::UpdateCurReqCnt - An error occurred attempting to read a response from the response queue. Verify that the translation server is running. Check errno for more information.
44	CAsyncTranslator::SetFileName - A call was made to send a SET command to the translation server, but this is the server.

C++ and Java API return codes

Code	Definition
45	CAsyncTranslator::StartTranslator - A call was made to send start commands to the translation server, but this is the translation server.
46	CAsyncTranslator::StartTranslator - The router type specified in the CDIRRequest is invalid. Valid types are file, pipe, socket, and email.
47	CAsyncTranslator::StartTranslator - The unit of work specified in the CDIRRequest is invalid. Valid types are envelope, transaction, and no commit.
48	CAsyncTranslator::Terminate - A call was made to send a terminate message to the translation server, but this is the translation server.
49	CAsyncTranslator::Terminate - A call was made to terminate the translator, but the translator has not been initialized yet.
50	CRemoteTranslator::Initialize - An attempt was made to initialize a remote translator that has already been initialized.
51	CRemoteTranslator::Initialize - An attempt was made to resolve the host name and an error occurred. Most likely, the host was not found. Verify the host name is correct and accessible from the translation system. Check errno for more information.
52	CRemoteTranslator::Initialize - Failed to create the socket to communicate with the translation server. Check errno for more information.
53	CRemoteTranslator::Initialize - Failed to connect to the remote translator. Verify that the remote translator daemon is running and that the host name is correct. Check errno for more information.

Appendix B. Mapping the MQRFH2 header to the JMS API

This appendix explains how the MQRFH2 mapping to the Java Message Service (JMS) API works when you are communicating with a JMS Client.

The MQSeries Java Message Service (JMS) implementation uses the Message Content Descriptor (MCD) folder of the MQRFH2 to carry information about the message, as described in the MQRFH2 header. By default, the Message Domain (MSD) property is used to identify whether the message is a text, bytes, stream, map, or object message. This value is set depending on the type of the JMS messages.

If the application calls `setJMSType`, it can set the MCD type field to a value of its choosing. This type field can be read by the MQSeries Integrator message flow, and a receiving JMS application can use the `getJMSType` method to retrieve its value. This applies to all kinds of JMS messages.

When a JMS application creates a text or bytes message, the application can set MCD folder fields explicitly by calling the `setJMSType` method and passing in a string argument in a special Universal Resource Identifier (URI) format as follows:

```
mcd://domain/[set]/[type][?format=fmt]
```

This URI form allows an application to set the MCD to a domain that is not one of the standard `jms_XXXX` values; for example, to domain `mrm`. It also allows the application to set any or all of the MCD `set`, `type`, and `format` fields if desired.

The string argument to `setJMSType` is interpreted as follows:

1. If the string does not appear to be in the special URI format (for example, it does not start with `mcd://`), then the string is added to the MCD folder as the type field.
2. If the string does start with `mcd://`, conforms to the URI format, and the message is a Text or Bytes message, then the URI string is split into its constituent parts. The domain part overrides the `jms_text` or `jms_bytes` value that would otherwise have been generated, and the remaining parts (if present) are used to set the `set`, `type`, and `format` fields in the MCD. Note that `set`, `type`, and `format` are all optional.

Mapping the MQRFH2 header to the JMS API

3. If the string starts with `mcd://` and the message is a Map, Stream or Object message, then the `setJMSType` call throws an exception. You cannot override the domain, or provide a set or format for these classes of messages, but you can provide a type if you wish.

When an MQ message is received with an MSD other than one of the standard `jms_xxxx` values, it is instantiated as a JMS text or bytes message and a URI-style JMSType is assigned to it. The receiving application can read this using the `getJMSType` method.

Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or

Notices

implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of

performance, compatibility or any other claims related to non-IBM products.

Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries, or both:

AIX
DB2
ExpEDIte
IBM
MQSeries
MVS
WebSphere

Java and all Java-related trademarks are trademarks of Sun Microsystems, Inc. in the United States, or other countries, or both.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company product and service names may be trademarks or services marks of others.

Notices

Index

A

- Adapter user exits 23
- API
 - example 33
 - return codes 32
- auditing 2

C

- C++ and Java API return codes 51
- calling from a C++ program 25
- command chaining 43
- comments 6
- configure database connections 15
- connections to server databases 14

E

- EDI standards support 3
- elements of the C++ API 25

F

- features 1

G

- glossary 6

I

- IBM WebSphere Data Interchange for Multiplatforms 1
- installing
 - Client 11
 - DB2 Connect 14
 - Server 6

J

- JMS implementation 57

M

- MCD 39
- MCD profiles
 - creating 41
- Message Content Descriptor 39
- MQRFH2 57
- MQRFH2 header 41

R

- related books 5
- reporting 2
- requirements
 - Client hardware 11
 - Client software 11
 - Server hardware 5
 - Server software 5
- return codes 51
- running from the command line 19
- running WebSphere Data Interchange Server
 - from the command line 19
 - triggering from an MQSeries Queue 21

S

- Service profile
 - creating 45
 - name 43
- setting up databases 8

T

- triggering from an MQSeries Queue 21



Product Number: 5724-C50