File Adapter for z/OS sequential files
Version 4.0

4 Feb 2008

Alexander Russell

Lakshman Yatawara

Vicente Suarez

IBM UK Ltd
Hursley Park
Winchester, Hampshire
United Kingdom

Take Note!

Before using this report be sure to read the general information under "Notices".

Fourth Edition, Feb 2008

This edition applies to Version 4.0 of the File Adapter plug-in nodes and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of contents

Table of figures

Notices

Note: The provisions set out in the following two paragraphs do not apply in any country, to the extent only that in such provision is inconsistent with mandatory local law.

The information contained, and any technique described, in this document has not been submitted to any formal IBM test and is distributed "AS-IS". The use of such information or technique (including any implementation of any such information or technique) is the user's responsibility and depends on the user's ability to evaluate and integrate them into the user's operational environment. While IBM has reviewed each item for accuracy in a specific situation, IBM offers no guarantee that the same or similar results will be obtained elsewhere. Users attempting to adapt any technique described in this document to their own environments do so at their own risk.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY, SATISFACTORY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states and countries do not allow the disclaimer of express or implied warranties in certain transactions, therefore the above statements may not apply to you. In any such event, you will only benefit from the minimum applicable implied warranty.

Any reference in this publication to an IBM product, program or service does not imply that IBM intends to make it available in all countries in which IBM operates.

Any reference in this publication to an IBM licensed program or another IBM product is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe applicable intellectual property rights may be used instead of the IBM licensed program or other IBM product.

The user is responsible for evaluating and verifying the operation of the material supplied in conjunction with this document in conjunction with other products, except those expressly designated by IBM.

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

The IBM Director of Licensing, International Business Machines Corporation, North Castle Drive, Armonk, New York 10504-1785 USA.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- MQSeries
- WebSphere MQ (WMQ)
- MQSI
- WebSphere MQ Integrator (WMQI)
- WebSphere Business Integration Message Broker (WBIMB)
- WebSphere Message Broker (WMB)

The following terms are trademarks of other companies:

- Windows NT/2000/XP

Acknowledgments

The following persons contributed to the design, development, testing and debugging of this SupportPac and without their help it would not be possible to complete this FileAdapter:

- Ralph Bateman – WebSphere Message Broker Change Team
- Peter Crocker – WebSphere Message Broker Development
- John Hosie - WebSphere Message Broker Development
- Anthony J. O'Dowd - WebSphere Broker Development
- Colin Paice – WebSphere Message Broker Performance Team
- Gary Willoughby – WebSphere Business Integration Lab Services manager

Summary of amendments

Date	Changes
30 May 2003	Initial release – Support for WMQI V2.1
25 February 2004	Version 2.0 release – Support for WBIMB V5
28 October 2005	Version 3.0 release – Support for WMB V6
26 May 2006	Version 3.1 – Add warnings about high memory usage
4 Feb 2008	Version 4.0 – Add support for WMB 6.1 and remove references to WMQI 2.1 and WBIMB V6.

Preface

The IA11 SupportPac can be used to integrate batch applications using WebSphere Message Broker (WMB V6.0 or V6.1) message flows. The File Adapter plug-in nodes add the capability of reading and writing messages from and to z/OS sequential files (QSAM data sets).

The File Adapter plug-in nodes are designed to be used in WMB message flows that represent the following scenarios:

1. Read records from an input file, propagate the records as messages, transform the messages and write them to one or more output files.

2. Read records from an input file, propagate the records as messages, transform the messages and put them on WMQ queues.

3. Get WMQ messages from a queue, transform the messages and write them to one or more output files.

The File Adapter has the following plug-in nodes (or user-defined nodes):

1. File Read plug-in node. To read records from a file and propagate one message per record to the output (“out”) terminal.
2. File Write plug-in node. To receive messages from the input (“in”) terminal and store them in an output file.
3. File Delete plug-in node. To receive a control message with a filename and delete the file.
4. File Rename plug-in node. To receive a control message with old and new file names and rename the file.

XML action control messages are used to indicate when to open, close, delete or rename files.

Bibliography

- *IBM WebSphere Message Broker Version 6.0 Information Center*.
- *IBM WebSphere Message Broker Version 6.0 User-Defined Extensions (messagebroker_User-defined_Extensions.pdf).*
- *IBM WebSphere Message Broker Version 6.1 Information Center*.
- *z/OS DFSMS: Using Data Sets,* IBM Corporation, SC26-7410.
- *z/OS MVS Programming: Authorized Assembler Services Guide,* IBM Corporation, SA22-7608.

Chapter 2. Overview

This SupportPac enables the integration of z/OS batch applications using WebSphere Message Broker (WMB V6.0 or V6.1). This SupportPac can be used while z/OS batch applications are replaced by messaging applications. It is expected that message routing and transformation business logic can be shared between z/OS batch and messaging applications.

Assumptions

The File Adapter plug-in nodes were developed under the following assumptions:

- The File Adapter plug-in nodes process only z/OS QSAM data sets.
- The supported record formats are:
  - fixed record length (F),
  - fixed blocked record length (FB),
  - variable record length (V) and,
  - variable blocked record length (VB).
- The File Adapter plug-in nodes process one message per record.
- The File Adapter provides the following plug-in nodes:
  - File Read plug-in node to read messages from a file,
  - File Write plug-in node to write messages to a file,
  - File Delete plug-in node to delete a file,
  - File Rename plug-in node to rename a file.
- It is possible to use each of the File Adapter plug-in nodes on its own within a message broker message flow.
- The File Adapter plug-in nodes **cannot** be used in multi-threaded message flows (flows deployed with the property “Additional Instances” set to a value different to zero).
- The File Adapter plug-in nodes have restrictions when used within a single transaction (Unit of Work). Please refer to each of the scenario descriptions later in this document.
- It is not possible to make changes to the message broker configuration (stop/start message flows or deploy message flows or deploy message sets or start/stop traces) while a file is open by a message flow.
- Updates to sequential files (QSAM data sets) cannot be part of any unit of work.

Scenarios

The File Adapter plug-in nodes are intended to be used in message flows that represent the following scenarios:

1. One input file to one or more output files.
2. One input file to WebSphere MQ (WMQ) messages.
3. WMQ messages to one or more output files.

One input file to one or more output files

Message flows in this scenario read records from a single input file, the record is encapsulated in a message, and these messages can then be transformed by compute nodes and written into one or more output files.

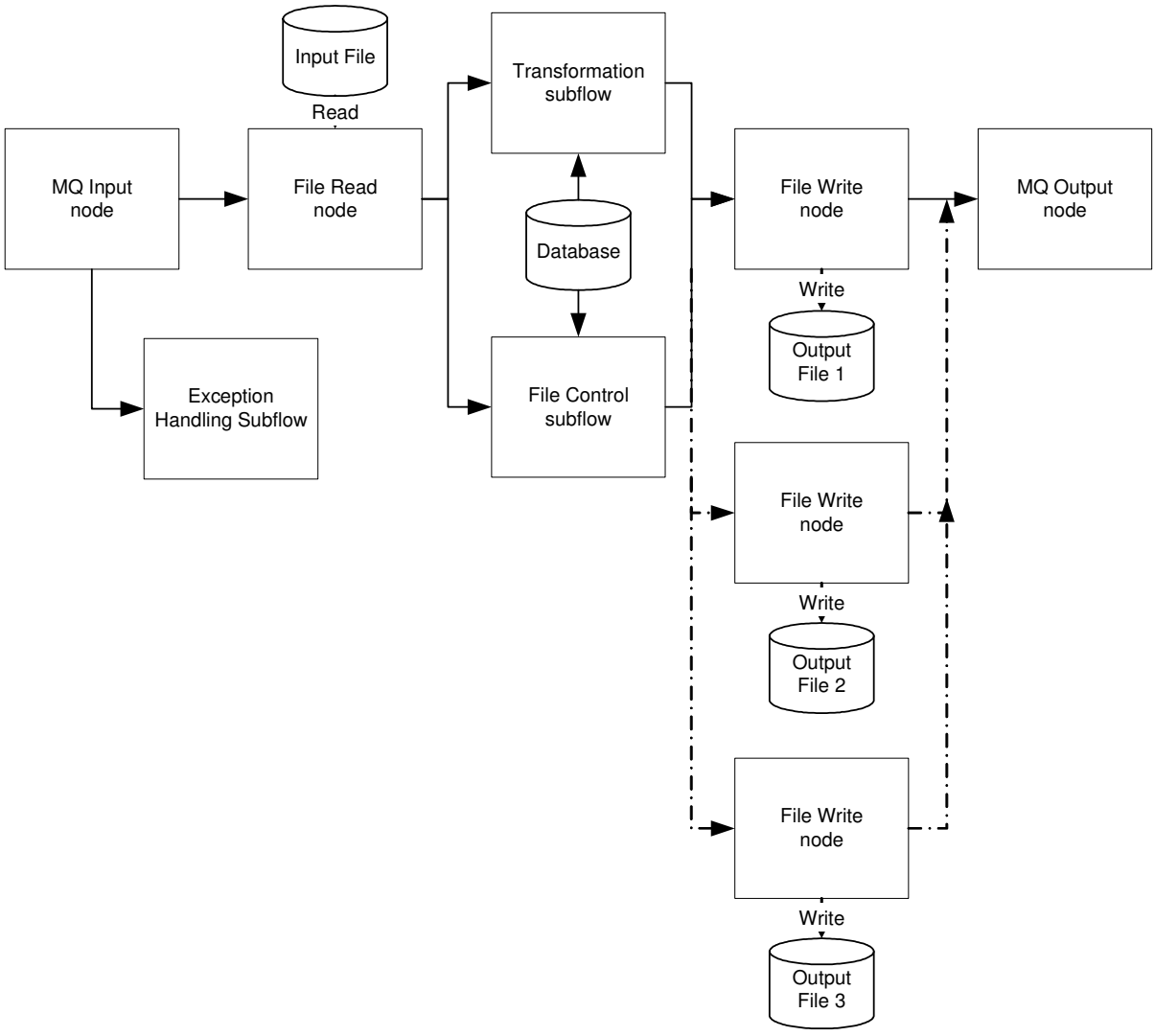

Figure 1 - File to file scenario

In this scenario it is possible to process all the messages in an input file as part of a single unit of work. The number of database updates within a single unit of work is limited. The limitation is the amount of resources used in the database manager and the time that would take to rollback in case of failure. It is recommended that a unit of work should not have more than 10,000 database updates within a single unit of work. This essentially limits the size of the files that can be processed within a single unit of work. In this scenario, if there are no database updates or inserts then it is possible to process large input files within a single unit of work because the resources that are coordinated are the MQGETs and MQPUTs which normally will be one MQGET for an input message and one MQPUT for an output message per file. In case of failure or error, the input message is put back on the input queue and the input message process can be retried.

One Input File to WMQ Messages

Message flows in this scenario read messages from an input file; they are transformed and put in WMQ queues.

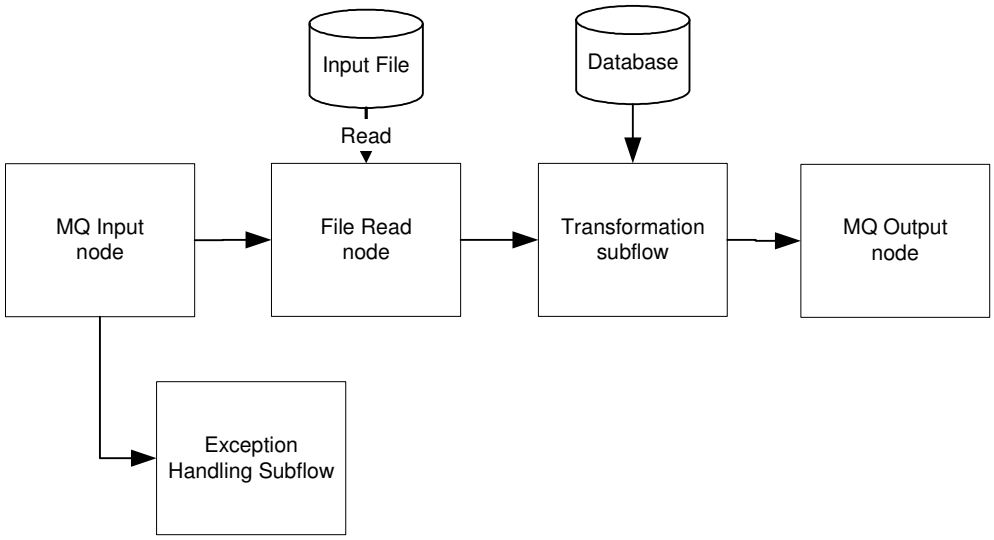

Figure 2 - Input file to WMQ messages scenario

In this scenario it might not be possible to process all the messages in an input file as part of a single unit of work. The number of database updates and/or uncommitted WMQ messages on a queue within a single unit of work is limited. The restriction is the amount of resources used in the database manager and in the queue manager to log all these updates and the time that would take to rollback in case of failure. It is recommended that a unit of work should not have more than 10,000 database updates and/or 10,000 uncommitted WMQ messages on a queue. This essentially limits the size of the files that can be processed within a single unit of work. Using a single unit of work inhibits the scalability of the message flow.

WMQ messages to one or more output files

Message flows in this scenario receive input messages on a WMQ queue, transform the messages and pass them to a File Write Node for storing in a file. The output file is controlled in one of two ways:

1. Passing action messages to the File Write plug-in node to open and close the file.
2. Using the properties of the File Write Node to specify the criteria, which determine, when to open and close the output file. The File Write Node will generate unique file names appending a suffix to the file name prefix. This prefix is specified in the node properties.

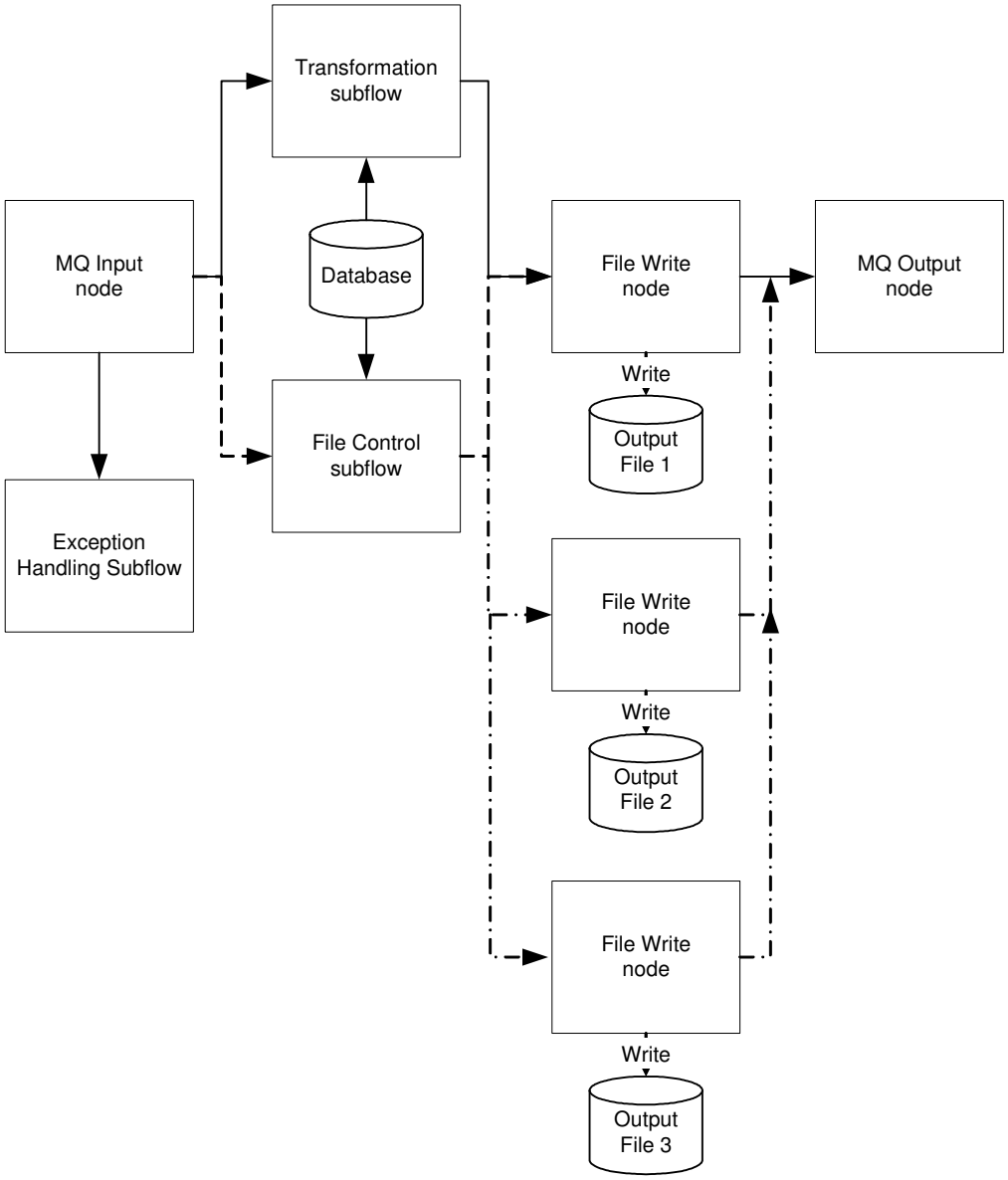

Figure 3 - WMQ messages to file scenario

In this scenario, each input message can be processed within a unit of work. In case of failure or error the output file must be preserved because it contains messages that cannot be recovered by WMQ.

WARNING - How to avoid allocating increasing amounts of memory

There is a potential problem allocating increasing amounts of memory that is only released at the end of the message flow. The message broker can run out of memory in the address space before completing the processing of the input file. This problem can affect message flows that implement the "File to File" or "File to Message" scenarios.

This problem is caused when the ESQL/Java code implementing transformation logic creates elements in the "Environment Tree" that are not deleted when they are not longer needed. To avoid the problem, it is recommended to delete the elements from the Environment Tree that are not required to process any subsequent records and before passing control to the FileWrite node or MQOutput node at the end of the process of each record.

If you use the following ESQL statement to set a variable in the environment:

SET Environment.MyVariables.Variable1 = MyValue1;

then use the following ESQL statement to delete the variable when it is not longer needed:

DELETE FIELD Environment.MyVariables.Variable1;

Chapter 3. Installing the plug-in nodes

This chapter describes how to install and configure the File Adapter plug-in nodes.

SupportPac contents

The following "zip" files can be downloaded from the internet with the contents of the SupportPac.

- The file named "**ia11_mb_v6.zip**" is used to install the SupportPac on WMB V6.0.
- The file named "**ia11_mb_v6.1.zip**" is used to install the SupportPac on WMB V6.1

The supplied "zip" file should be unzipped into a temporary (Windows 2000/XP) directory. The following sub-directories are created:

WMB V6.0/V6.1

- FileAdapter\ (miscellaneous files including this document (.pdf))
- FileAdapterTest_MessageSet (test message set used with some test flows)
- FileAdapter\FileAdapterTestFlows\ (test message flows)
- FileAdapter\lil (executable plug-in node for WMB for z/OS broker)
- FileAdapter\EclipseInstall (Eclipse Update site)
- FileAdapter\Licenses (IBM Software license aggreement)

Prerequisites

You can install the FileAdapter on the following versions of WebSphere Business Integration Message Broker:

- WebSphere Message Broker (WMB) for z/OS Version 6.0 or Version 6.1 GA and above.

Supported Platforms

This SupportPac has been developed and tested in the following z/OS environments:

- WMB V6.0 in z/OS V1.5.
- WMB V6.1 in z/OS V1.7.

Installing the plug-in nodes

This section describes how to install the FileAdapter nodes. The installation process for the supported versions of Message Broker is described in:

- "Installation process for WMB V6.0" on page 6.
- "Installation process for WMB V6.1" on page 7.

Installation process for WMB V6.0

1. Download SupportPac IA11 for WMB V6.
2. For WMB V6.0 unzip "**ia11_mb_v6.zip"** onto your local file system, preferable your **C:** drive.
3. The installation instructions assume that:
   - You placed the extracted files in your **C:** drive.
   - WMB toolkit is installed in **C:\WMB\MessageBrokersToolkits\6.0**.

Installation of the FileAdapter plug-ins in the Message Broker Toolkit

Use the Message Broker Toolkit to install the FileAdapter nodes. To do this:

1. Start the Message Broker Toolkit Workbench.
2. Select **Help -> Software Updates -> Find and Install...**
3. From the **Features Updates** pane, select **Search for new features to install** → next.
4. From the **Update Sites to visit** pane, click **New Local Site.**
5. From the **Browse for Files** pane, find the following folder: **C:\FileAdapter\EclipseInstall** → OK.
6. Select and expand the **EclipseInstall** directory. Check the **IBM WebSphere Message Broker File Adapter SupportPac**. Click next.
7. From the **Search Results** pane. The following feature is displayed: **SupportPac IA11 – File Adapter nodes**. Check this feature and click next.
8. Accept the terms in the license agreements. Click next.
9. From the **Install Location** pane. Select the following available site: **<WMB_INSTALL_DIR>\MessageBrokersToolkits\6.0\eclipse** and click finish.
10. Click **Install** to start the feature installation process.
11. Select **Yes** to restart the workbench when the installation is completed.

Installation process for WMB V6.1

1. Download SupportPac IA11 for WMB V6.
2. For WMB V6.1 unzip "**ia11_mb_v6.1.zip"** onto your local file system, preferable your **C:**\ drive.
3. The installation instructions assume that:
   - You placed the extracted files in your **C:**\ drive.
   - WMB toolkit is installed in **C:\Program Files\IBM\WMBT610**.

Installation of the FileAdapter plug-ins in the Message Broker Toolkit

Use the Message Broker Toolkit to install the FileAdapter nodes. To do this:

1. Start the Message Broker Toolkit Workbench.
2. Select **Help -> Software Updates -> Find and Install...**
3. From the **Features Updates** pane, select **Search for new features to install** → next.
4. From the **Update Sites to visit** pane, click **New Local Site.**
5. From the **Browse for Files** pane, find the following folder: **C:\FileAdapter\EclipseInstall** → OK.
6. Click OK on the Edit Local Site window.
7. Check that the FileAdapter/EclipseInstall site is selected. Click Finish.
8. From the **Search Results** window. Select and expand the **EclipseInstall** directory. Select the **SupportPac IA11 – File Adapter Nodes 4.0.0**. Click next.
9. Accept the terms in the license agreements. Click next.
10. From the **Installation** pane. Check that the feature **SupportPac IA11 – File Adapter Nodes 4.0.0** has the following installation directory associated: **C:\Program Files\IBM\WMBT610** and click finish.
11. Click **Install** to start the feature installation process on the **Feature Verification** window.
12. Select **Yes** to restart the workbench when the installation is completed.

z/OS Broker installation

The following is assumed in the instructions that follow:

- The WMB for z/OS installation path is **/usr/lpp/mqsi**
- The broker name is **WI01BRK**
- The broker directory is **/wmqi/WI01BRK**.
- The plug-in executable (lil file) directory is **/wmqi/WI01BRK/lil**.

Create the plug-in directory

On z/OS go to the Unix System Services (USS) shell (**omvs** from ISPF option 6):

```
/u/wi01brk:>cd /wmqi/WI01BRK

/wmqi/WI01BRK:>mkdir lil
```

Install the plug-in executable

On Windows use FTP to copy the lil file:

```
C:\FileAdapter>cd lil

C:\FileAdapter\lil>ftp HURMVS99
Connected to HURMVS99.hursley.ibm.com.
220-FTPD1 IBM FTP CS V2R10 at HURMVS99.HURSLEY.IBM.COM, 10:06:25 on
2003-02-21.
220 Connection will close if idle for more than 5 minutes.
User (HURMVS99.hursley.ibm.com:(none)): wi01brk
331 Send password please.
Password:
230 WI01BRK is logged on.  Working directory is "WI01BRK.".
ftp> cd /wmqi/WI01BRK/lil
250 HFS directory /wmqi/WI01BRK/lil is the current working directory
ftp> bin
200 Representation type is Image
ftp> put FileAdapter.lil
200 Port request OK.
125 Storing data set /wmqi/WI01BRK/lil/FileAdapter.lil
250 Transfer completed successfully.
ftp: 716800 bytes sent in 0.94Seconds 761.74Kbytes/sec.
ftp> bye
221 Quit command received. Goodbye.

C:\FileAdapter\lil>
```

Note: If it is necessary to locate the “FileAdapter.lil” in a path that is shared by all brokers then copy the lil file to “/usr/lpp/mqsi/lil”.

Set the lil file permissions

On z/OS (USS):

```
/wmqi/WI01BRK/lil:>ls -l
total 1400
-rw-r-----   1 WI01BRK  HSTGRP    716800 Feb 21 10:07
FileAdapter.lil
/wmqi/WI01BRK/lil:>chmod 755 FileAdapter.lil
/wmqi/WI01BRK/lil:>ls -l
total 1400
-rwxr-xr-x   1 WI01BRK  HSTGRP    716800 Feb 21 10:07
FileAdapter.lil
/wmqi/WI01BRK/lil:>
```

Note: you may use a more restrictive set of permissions (for example, **750**) if they work in your installation.

Check the “FileAdapter.lil” owner and group because one of them should give the correct access permissions (read+execute) to the broker user id (WI01BRK). Change owner or group as appropriate.

Update the broker lil path

If the lil path "/wmqi/WI01BRK/lil" is already defined in the LILPATH variable in ENVFILE or there is a need to place the "FileAdapter.lil" in the common lil path ("/usr/lpp/mqsi/lil") to all the brokers then skip this section.

WMB

On z/OS (TSO/ISPF):

Assume that the broker's customization PDSE library is "**WMB.WI01BRK.JCL**".

Stop the broker, if it is running.

Edit the WMB.WI01BRK.JCL(BIPBPROF):

Update the following statement as follows:

Change
export MQSI_LILPATH=£MQSI_FILEPATH/lil
to
export MQSI_LILPATH=£MQSI_FILEPATH/lil:/wmqi/WI01BRK/lil (for WMB V6.0)
or
export MQSI_LILPATH32=£MQSI_FILEPATH/lil:/wmqi/WI01BRK/lil (for WMB V6.1)

Execute job WMB.WI01BRK.JCL(BIPGEN) to recreate the broker ENVFILE.

Start the broker.

This completes the installation of the File Adapter plug-in nodes.

Chapter 4. Using the plug-in nodes

File Read plug-in

Description

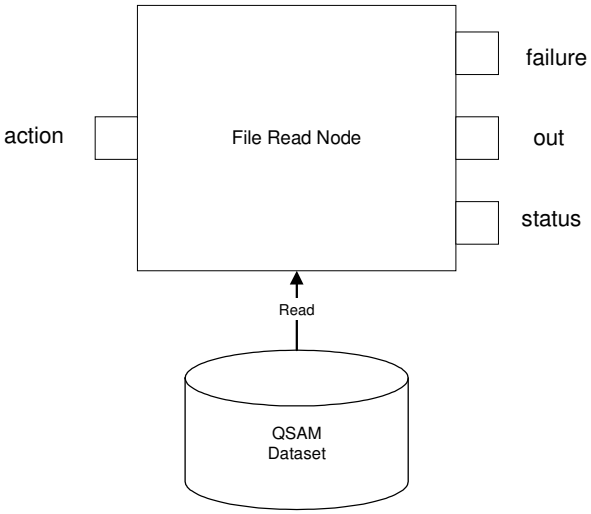

Figure 4 - File Read

The File Read plug-in reads from a sequential file (QSAM data set in z/OS) and propagates each record as a message through the “out” terminal. The input file process is triggered when an “open input file” action message arrives at the “action” terminal.

The File Read plug-in validates the action message, opens the required input file (opened input file status message is propagated to the “status” terminal), reads records from the file and for each record it propagates a BLOB data message to the “out” terminal.

If data messages are propagated directly to an MQOutput node, it is necessary to add a Reset Content Descriptor node (RCD) (BLOB to BLOB) to force the message flow to make a copy of the data message before propagating it to the MQOutput node. If the RCD node is not included, the File Read node generates an error exception (“Invalid state for finalize” from cniFinalize()) and terminates the file process. If the data messages are propagated to a transformation subflow, the RCD node is not required.

When the end of file is detected, the file is closed and the process is terminated and a “closed input file” status message is propagated to the “status” terminal.

If an error condition is detected, an exception message is propagated to the “failure” terminal or if the “failure” terminal is not wired, then a user exception is thrown to terminate the processing. After an exception is detected the input file is closed. An exception message is propagated to the “failure” terminal instead of a “closed input file” status message.

Copies of action, status and exception messages are maintained in the Environment tree of the message flow.

Input

Node Properties

- [D] Default.DebugNode: None, Performance or Full (None is default). The “Performance” option generates data to monitor the duration of key activities during the FileRead process. The “Full” option generates debugging and trace information as well as the performance data.
- [D] Default.TraceFilename: string (filename). File to write the debugging and trace information, if TraceFilename is not specified (blank) then the debugging and trace information is written to the broker’s ‘stdout’.
- [D] Default.Encoding: integer (WMQ encoding value. Default value is 785). This value represents the encoding of the numeric fields in the records of the input file. This is a numeric code that describes how the integer, decimal and floating point fields are encoded. Refer to the WebSphere MQ documentation for more information.
- [D] Default.CCSID: integer (Default value is 500). This coded character set identifier (CCSID) is used to decode the text data in the records of the input file.
- [D] Default.AddMQMDToBLOBMessage: True/False (False is default). If “True”, File Read appends an MQMD header to each BLOB data message propagated with each record read from the input file.
  Note: It is not necessary to add the MQMD header to the BLOB message when the message is not being propagated to an MQOutput node (or similar). There is a performance gain when the messages are propagated to File Write nodes without the MQMD header.

[D] = Optional but default value exists.

Action terminal

Note: Refer to Appendix A for a description of the elements of the action messages.

Open Input File Action Message
The Properties and MQMD elements can be present with the following message in XML format:

```
[M]<FileAdapter>
[M]             <Filename>’string’
[M]                     <Action>OPEN</Action>
[M]                     <Type>INPUT</Type>
[M]                     <Attributes>
[M]                             <Disposition>
[M]                                     <Status>string</Status>
[O]                                     <Normal>string</Normal>
[O]                                     <Conditional>string</Conditional>
[M]                             </Disposition>
[M]                     </Attributes>
[M]             </Filename>
[M]</FileAdapter>
```

Note: [M] is a mandatory element and [O] is an optional element.

Local Environment

No data is expected in the Local Environment tree.

Environment

No data is expected in the Environment tree.

Exception List

No data is expected in the Exception List tree.

Input File

The input file must be an existing QSAM data set with record format of fixed (F), Variable (V), Fixed Blocked (FB), or Variable Blocked (VB). The file can be open with disposition OLD (exclusive use) or SHR (shared use).

Output

Note: Refer to Appendix A for a description of the elements of the action, status and exception messages.

Status Terminal

It is recommended that when messages propagated of the Status terminal and passed to a MQOutput node the following attribute of the node is changed: **MQOutput → Properties → Advanced → Message Context → Default.**

Opened Input File status message
The Properties and MQMD elements are created (their content is copied from the input message if they exist) together with the following message in XML format:

```
<FileAdapter>
        <Filename>'string'
                <Status>OPENED</Status>
                <Type>INPUT</Type>
                <Attributes>
                        <Disposition>
                                <Status>string</Status>
                                <Normal>string</Normal>
                                <Conditional>string</Conditional>
                        </Disposition>
                                <RecordFormat>string</RecordFormat>
                                <RecordLength>integer</RecordLength>
                                <BlockSize>integer</BlockSize>

                </Attributes>
                <TimeFileOpened> timestamp</TimeFileOpened>
        </Filename>
</FileAdapter>
```

Closed Input File status
The Properties and MQMD elements are created (their content is copied from the input message if they exist) together with the following message in XML format:

```
<FileAdapter>
        <Filename>'string'
                <Status>CLOSED</Status>
                <Type>INPUT</Type>
                <Attributes>
                        <Disposition>
                                <Status>string</Status>
                                <Normal>string</Normal>
                                <Conditional>string</Conditional>
```



```
                    </Disposition>
                        <RecordFormat>string</RecordFormat>
                        <RecordLength>integer</RecordLength>
                        <BlockSize>integer</BlockSize>

                </Attributes>
                <TimeFileOpened> timestamp</ TimeFileOpened>
                <TimeFileClosed> timestamp</ TimeFileClosed>
                <RecordsProcessed>integer</RecordsProcessed>
        </Filename>
</FileAdapter>
```

Out Terminal

Data messages are propagated to the “out” terminal. The Properties (and MQMD if specified by the AddMQMDToBLOBMessage node property) elements are included in the data message. The data message is in BLOB format. The following is an example of internal message tree:

```
(
  (0x1000000)Properties = (
    (0x3000000)MessageSet      = ''
    (0x3000000)MessageType     = ''
    (0x3000000)MessageFormat   = ''
    (0x3000000)Encoding        = 785
    (0x3000000)CodedCharSetId  = 500
    (0x3000000)Transactional   = TRUE
    (0x3000000)Persistence     = TRUE
    (0x3000000)CreationTime    = GMTTIMESTAMP '2002-12-09 14:46:25.129'
    (0x3000000)ExpirationTime  = -1
    (0x3000000)Priority        = 0
    (0x3000000)ReplyIdentifier =
X'000000000000000000000000000000000000000000000000'
    (0x3000000)ReplyProtocol   = 'MQ'
    (0x3000000)Topic           = NULL
  )
  (0x1000000)MQMD       = (
    (0x3000000)SourceQueue      = 'FILE.READ.ACTION_QUEUE'
    (0x3000000)Transactional    = TRUE
    (0x3000000)Encoding         = 785
    (0x3000000)CodedCharSetId   = 500
    (0x3000000)Format           = 'MQSTR   '
    (0x3000000)Version          = 2
    (0x3000000)Report           = 0
    (0x3000000)MsgType          = 1
    (0x3000000)Expiry           = -1
    (0x3000000)Feedback         = 0
    (0x3000000)Priority         = 0
    (0x3000000)Persistence      = 1
    (0x3000000)MsgId            =
X'414d51204d515349514d2020202020200c8ff43d20002101'
    (0x3000000)CorrelId         =
X'000000000000000000000000000000000000000000000000'
    (0x3000000)BackoutCount     = 0
    (0x3000000)ReplyToQ         = 'MQREPLY
'
    (0x3000000)ReplyToQMgr      = 'MYQM
'
    (0x3000000)UserIdentifier   = 'db2admin    '
    (0x3000000)AccountingToken  =
X'16010515000000be4ed5348c77292b6a5cdb38ee0300000000000000000000b'
    (0x3000000)ApplIdentityData = '                                '
```



```
    (0x3000000)PutApplType      = 11
    (0x3000000)PutApplName      = '\examples\plugin\mqsiput.exe'
    (0x3000000)PutDate          = DATE '2002-12-09'
    (0x3000000)PutTime          = GMTTIME '14:46:26.160'
    (0x3000000)ApplOriginData   = '    '
    (0x3000000)GroupId          =
X'000000000000000000000000000000000000000000000000'
    (0x3000000)MsgSeqNumber     = 1
    (0x3000000)Offset           = 0
    (0x3000000)MsgFlags         = 0
    (0x3000000)OriginalLength   = -1
  )
  (0x1000000)BLOB       = (
    (0x3000000)UnknownParserName = 'MQSTR'
    (0x3000000)BLOB              =
X'48656c6c6f20576f726c64210a486f772061726520796f7520746f6461793f0a4920616d2
0666965207468612e'
  )
)
```

Failure Terminal

An exception message is propagated to the “failure” terminal when an error exception is detected by the File Read plug-in node. The Properties and MQMD elements are created (their content is copied from the input message if they exist) together with the following messages in XML format:

Exception when the action message is invalid

```
<FileAdapter>
        <Exception>FileRead</Exception>
        <ErrorText>string</ErrorText>
        <TimeException>timestamp</TimeException>
</FileAdapter>
```

Exception when the dynalloc function returns errors

```
<FileAdapter>
        <Filename>‘string’
                <Exception>FileRead</Exception>
                <Function>dynalloc</Function>
                <ErrorCode>integer</ErrorCode>
                <InformationCode>integer</InformationCode>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```

Exception when system functions return errors

```
<FileAdapter>
        <Filename>’string’
                <Exception>FileRead</Exception>
                <Function>string</Function>
                <ErrorNumber>integer</ErrorNumber>
                <ErrorText>string</ErrorText>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```

Local Environment

No data is created in the Local Environment tree.

Environment

The Environment tree is used to keep the status of the file process. Copies of action, status and exception messages are added to the environment tree. The following is the tree structure:

Environment.FileAdapter.FileRead: string (Filename)

Environment.FileAdapter.FileRead.ActionMessage
Environment.FileAdapter.FileRead.ActionMessage.Action: string (OPEN)
Environment.FileAdapter.FileRead.ActionMessage.Type: string (INPUT)
Environment.FileAdapter.FileRead.ActionMessage.Attributes
Environment.FileAdapter.FileRead.ActionMessage.Attributes.Disposition.Status: string (OLD or SHR)
Environment.FileAdapter.FileRead.ActionMessage.Attributes.Disposition.Normal: string (KEEP, DELETE, CATLG, UNCATLG)
Environment.FileAdapter.FileRead.ActionMessage.Attributes.Disposition.Conditional: string (KEEP, DELETE, CATLG, UNCATLG)

Environment.FileAdapter.FileRead.StatusMessage
Environment.FileAdapter.FileRead.StatusMessage.Status: string (OPENED, CLOSED)
Environment.FileAdapter.FileRead.StatusMessage.Type: string (INPUT)
Environment.FileAdapter.FileRead.StatusMessage.Attributes
Environment.FileAdapter.FileRead.StatusMessage.Attributes.RecordFormat: string (F, V, FB, or VB)
Environment.FileAdapter.FileRead.StatusMessage.Attributes.RecordLength: integer (bytes)
Environment.FileAdapter.FileRead.StatusMessage.Attributes.BlockSize: integer (bytes)
Environment.FileAdapter.FileRead.StatusMessage.RecordsProcessed: integer
Environment.FileAdapter.FileRead.StatusMessage.TimeFileOpened: timestamp
Environment.FileAdapter.FileRead.StatusMessage.TimeFileClosed: timestamp

Environment.FileAdapter.FileRead.ExceptionMessage
Environment.FileAdapter.FileRead.ExceptionMessage.Exception: String (FileRead)
Environment.FileAdapter.FileRead.ExceptionMessage.Function: String
Environment.FileAdapter.FileRead.ExceptionMessage.ErrorCode: integer
Environment.FileAdapter.FileRead.ExceptionMessage.InformationCode: string
Environment.FileAdapter.FileRead.ExceptionMessage.ErrorNumber: integer
Environment.FileAdapter.FileRead.ExceptionMessage.ErrorText: string
Environment.FileAdapter.FileRead.ExceptionMessage.TimeException: timestamp

Exception List

The broker creates the Exception List tree when an exception is thrown by the plug-in using the function *cciThrowException()* and the tree is populated with data provided in the function parameters.

File Write plug-in

Description

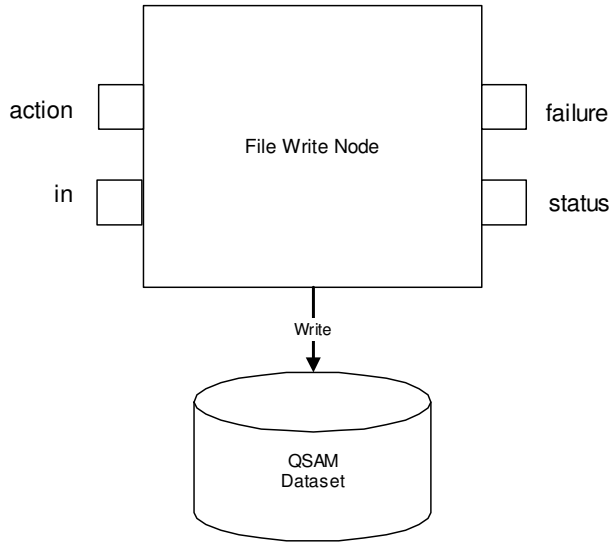

Figure 5 - File Write

The File Write plug-in node receives data messages on the “in” terminal and writes them as records in an output file (QSAM data set in z/OS). Status messages are propagated to the “status” terminal when an output file is opened or closed.

If an error condition is detected, an exception message is propagated to the “failure” terminal or if the “failure” terminal is not wired, then a user exception is thrown to terminate the message flow processing. After an exception is detected the output file is closed. An exception message is propagated to the “failure” terminal instead of a “closed output file” status message.

Copies of action and status messages are maintained in the Environment tree of the message flow.

The File Write plug-in node has two distinct modes of operation: The first one is called “Action Message Controlled Mode” and the second one is called “Automated Mode”.

Action Message Controlled Mode

The File Write plug-in node opens and closes the output file when it receives a valid (open/close/exceptionclose) action message on the “action” terminal. The life cycle of the output file is controlled by action messages.

Automated Mode

The File Write plug-in node properties define the file name, the file attributes and when to close the output file. A new file is opened when a data message arrives on the “in” terminal and there is no output file already opened.

The File Write plug-in node has the following algorithm to generate unique output file names:

> The Filename node property specifies a file name stem used to prefix any generated file name. Each time a new output file is required a unique suffix is appended to the file name stem. This suffix is formed by the year and Julian day (Dyyddd) followed by the time of day (Thhmmss) and a sequence number (Snnnn). For example, if the node property “Filename”

has the value of “MY.FILE” then the generated file name is MY.FILE.Dyyddd.Thhmmss.Snnnn.

The sequence number is initialized in zeros (0000) when a new file name is generated. This number is incremented if a duplicate file name is found and the open is retried with the same date and time but different sequence. If a sequence of 9999 is exceeded then a new name is generated again to get a different date and time.

The output file is closed when one of the following conditions is met:

1. The number of records written to file has exceed the MaxNumberRecords=integer (node property)
2. The number of minutes that output file has been opened has exceed the MaxOpenTime=minutes (node property)
3. The GMT (UTC) time of day is now TimeToClose=hhmm (node property)

When the File Write plug-in node is operating in Automated Mode and an action message is received on the “action” terminal, an error exception is thrown. Action messages are not allowed in Automated Mode. The “ExceptionClose” action message is allowed to close the output file and then throw an exception to terminate the processing.

Input

Node Properties

- [D] Default.DebugNode: None, Performance or Full (None is default). The “Performance” option generates data to monitor the duration of key activities during the FileWrite process. The “Full” option generates debugging and trace information as well as performance data.
- [D] Default.TraceFilename: string (filename). File to write the debugging and trace information, if TraceFilename is not specified (blank) then the debugging and trace information is written to the broker’s ‘stdout’.
- [D] Default.AutomatedMode: True/False (False is default). To define the plug-in mode of operation.
- [D] Default.MultithreadedFile: True/False (“False” is default). To define if the output file is shared by all threads (MultithreadedFile=True) or each thread has its own output file (MultithreadedFile=False).
- [M] Automated.TimerExpiration: integer (seconds). Internal timer expiration interval.
- [O*] Automated.MaxNumberRecords: integer. Maximum number of records to be written in each file. Default value is zero to indicate no limit in the number of records.
- [O*] Automated.MaxOpenTime: integer (minutes). Maximum open time in minutes. Default value is zero to indicate no time limit.
- [O*] Automated.TimeToClose: hhmm. GMT (UTC) Time of day to close the file.
- [M] Automated.Filename: string. File name stem for automated name generation.
- [D] Automated.DispositionStatus: NEW or MOD (NEW is default).
- [D] Automated.DispositionNormal: KEEP, CATLG, DELETE (CATLG is default).
- [D] Automated.DispositionConditional: KEEP, CATLG, DELETE, UNCATLG (CATLG is default).
- [D] Automated.RecordFormat: none, F, V, FB, VB (none is default).
- [O] Automated.RecordLength: integer (bytes)
- [O] Automated.BlockSize: integer (bytes)
- [D] Automated.SpaceAllocationUnit: none, TRK or CYL (node is default).
- [O] Automated.SpacePrimary: integer (Allocation units).
- [O] Automated.SpaceSecondary: integer (Allocation units)
- [O] Automated.Unit: string
- [O] Automated.DataClass: string
- [O] Automated.ManagementClass: string
- [O] Automated.StorageClass: string

[D] = Optional but default value exists
[M] = Mandatory. If AutomatedMode = “False” then all these properties are ignored
[O] = Optional
[O*] = At least one of these properties must be specified.

In Terminal

Data messages are received on the “in” terminal. The Properties and MQMD elements should be included in the data message. The data message can be in BLOB, XML or MRM formats. These data messages are written as records to the output file opened by the File Write plug-in node.

Action Terminal

Note: Refer to Appendix A for a description of the elements of the action messages.

Action messages to open and close the output file are expected on the “action” terminal when the File Write plug-in node is operating in “Action Message Controlled Mode”.

The “ExceptionClose” message can be used to force closing any open file and terminating the process with a user generated exception.

Open Output File action message
The Properties and MQMD elements are required together with the following message in XML format:

```
[M]<FileAdapter>
[M]             <Filename>string
[M]                     <Action>OPEN</Action>
[M]                     <Type>OUTPUT</Type>
[M]                     <Attributes>
[M]                             <Disposition>
[M]                                     <Status>string</Status>
[O]                                     <Normal>string</Normal>
[O]                                     <Conditional>string</Conditional>
[M]                             </Disposition>
[O*]                            <RecordFormat>string</RecordFormat>
[O*]                            <RecordLength>integer</RecordLength>
[O*]                            <BlockSize>integer</BlockSize>
[O]                             <Space>
[O*]                                    <AllocationUnit>string</AllocationUnit>
[O*]                                    <Primary>integer</Primary>
[O]                                     <Secondary>integer</Secondary>
[O]                             </Space>
[O*]                            <DataClass>string</DataClass>
[O]                             <ManagementClass>string</ManagementClass>
[O]                             <StorageClass>string</StorageClass>
[O]                             <Unit>string</Unit>
[M]                     </Attributes>
[M]             </Filename>
[M]</FileAdapter>
```

Notes: [M] is a mandatory element, [O] is an optional element and [O*] at least one of these properties must be specified.

Filename does not require single ‘quotes’ when used to open a new output file.

Close Output File action message
The Properties and MQMD elements are required together with the following message in XML format:

```
<FileAdapter>
```



```
        <Filename>'string'
                <Action>CLOSE</Action>
                <Type>OUTPUT</Type>
        </Filename>
</FileAdapter>
```

Exception Close Output File action message
The Properties and MQMD elements are required together with the following message in XML format:

```
<FileAdapter>
        <Filename>'*'
                <Action>EXCEPTIONCLOSE</Action>
                <Type>OUTPUT</Type>
        </Filename>
</FileAdapter>
```

Local Environment

No data is expected in the Local Environment tree.

Environment

No data is expected in the Environment tree

Exception List

No data is expected in the Exception List tree

Output

Note: Refer to Appendix A for a description of the elements of status and exception messages.

Status Terminal

It is recommended that when messages propagated of the Status terminal and passed to a MQOutput node the following attribute of the node is changed: **MQOutput → Properties → Advanced → Message Context → Default.**

Opened Output File status message
The Properties and MQMD elements are required together with the following message in XML format:

```
<FileAdapter>
        <Filename>string
                <Status>OPENED</Status>
                <Type>OUTPUT</Type>
                <Attributes>
                        <Disposition>
                                <Status>string</Status>
                                <Normal>string</Normal>
                                <Conditional>string</Conditional>
                        </Disposition>
                        <RecordFormat>string</RecordFormat>
                        <RecordLength>integer</RecordLength>
                        <BlockSize>integer</BlockSize>
                        <Space>
                                <AllocationUnit>string</AllocationUnit>
                                <Primary>integer</Primary>
                                <Secondary>integer</Secondary>
                        </Space>
```



```
                    <Unit>string</Unit>
                    <DataClass>string</DataClass>
                    <ManagementClass>string</ManagementClass>
                    <StorageClass>string</StorageClass>
                </Attributes>
                <TimeFileOpened> timestamp</TimeFileOpened>
        </Filename>
</FileAdapter>
```

Closed Output File status

The Properties and MQMD elements are required together with the following message in XML format:

```
<FileAdapter>
        <Filename>string
                <Status>CLOSED</Status>
                <Type>OUTPUT</Type>
                <Attributes>
                        <Disposition>
                                <Status>string</Status>
                                <Normal>string</Normal>
                                <Conditional>string</Conditional>
                        </Disposition>
                        <RecordFormat>string</RecordFormat>
                        <RecordLength>integer</RecordLength>
                        <BlockSize>integer</BlockSize>
                        <Space>
                                <AllocationUnit>string</AllocationUnit>
                                <Primary>integer</Primary>
                                <Secondary>integer</Secondary>
                        </Space>
                        <Unit>string</Unit>
                        <DataClass>string</DataClass>
                        <ManagementClass>string</ManagementClass>
                        <StorageClass>string</StorageClass>
                </Attributes>
                <TimeFileOpened> timestamp</ TimeFileOpened>
                <TimeFileClosed> timestamp</ TimeFileClosed>
                <RecordsProcessed>integer</RecordsProcessed>
                <RecordsTruncated>integer</RecordsTruncated>
        </Filename>
</FileAdapter>
```

Failure Terminal

An exception message is propagated to the “failure” terminal when an error exception is detected by the File Write plug-in node. The Properties and MQMD elements are required together with the following messages in XML format:

Exception when the action messages are invalid

```
<FileAdapter>
        <Exception>FileWrite</Exception>
        <ErrorText>string</ErrorText>
        <TimeException>timestamp</TimeException>
</FileAdapter>
```

Exception when the dynalloc function returns errors

```
<FileAdapter>
```



```
        <Filename>'string'
                <Exception>FileWrite</Exception>
                <Function>dynalloc</Function>
                <ErrorCode>integer</ErrorCode>
                <InformationCode>integer</InformationCode>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```

Exception when the other functions return errors

```
<FileAdapter>
        <Filename>'string'
                <Exception>FileWrite</Exception>
                <Function>string</Function>
                <ErrorNumber>integer</ErrorNumber>
                <ErrorText>string</ErrorText>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```

ExceptionClose

```
<FileAdapter>
        <Exception>FileWrite</Exception>
        <ErrorText>
        Closing all files on 'EXCEPTIONCLOSE' FileAdapter XML control message
        </ErrorText>
        <TimeException>timestamp</TimeException>
</FileAdapter>
```

Local Environment

No data is created in the Local Environment tree.

Environment

The Environment tree is used to keep the status of the file process. Copies of action, status and exception messages are added to the environment tree. The following is the tree structure:

Environment.FileAdapter.FileWrite: string (Filename)

Environment.FileAdapter.FileWrite.ActionMessage
Environment.FileAdapter.FileWrite.ActionMessage.Action: string (OPEN or CLOSE)
Environment.FileAdapter.FileWrite.ActionMessage.Type: string (OUTPUT)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Disposition.Status: string (NEW or MOD)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Disposition.Normal: string (KEEP, DELETE, CATLG, UNCATLG)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Disposition.Conditional: string (KEEP, DELETE, CATLG, UNCATLG)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.RecordFormat: string (F, V, FB or VB)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.RecordLength: integer (bytes)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.BlockSize: integer (bytes)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Space.AllocationUnit: string (CYL, TRK)
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Space.Primary: integer
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Space.Secondary: integer
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.Unit: string
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.DataClass: string
Environment.FileAdapter.FileWrite.ActionMessage.Attributes.ManagementClass: string

Environment.FileAdapter.FileWrite.ActionMessage.Attributes.StorageClass: string

Environment.FileAdapter.FileWrite.StatusMessage
Environment.FileAdapter.FileWrite.StatusMessage.Status: string (OPENED, CLOSED)
Environment.FileAdapter.FileWrite.StatusMessage.Type: string (OUTPUT)
Environment.FileAdapter.FileWrite.StatusMessage.Attributes
Environment.FileAdapter.FileWrite.StatusMessage.Attributes.*(Refer to ActionMessage)*
Environment.FileAdapter.FileWrite.StatusMessage.RecordsProcessed: integer
Environment.FileAdapter.FileWrite.StatusMessage.RecordsTruncated: integer
Environment.FileAdapter.FileWrite.StatusMessage.TimeFileOpened: timestamp
Environment.FileAdapter.FileWrite.StatusMessage.TimeFileClosed: timestamp

Environment.FileAdapter.FileWrite.ExceptionMessage
Environment.FileAdapter.FileWrite.ExceptionMessage.Exception: String (FileWrite)
Environment.FileAdapter.FileWrite.ExceptionMessage.Function: String
Environment.FileAdapter.FileWrite.ExceptionMessage.ErrorCode: integer
Environment.FileAdapter.FileWrite.ExceptionMessage.InformationCode: integer
Environment.FileAdapter.FileWrite.ExceptionMessage.ErrorNumber: integer
Environment.FileAdapter.FileWrite.ExceptionMessage.ErrorText: string
Environment.FileAdapter.FileWrite.ExceptionMessage.TimeException: timestamp

Exception List

The broker creates the Exception List tree when an exception is thrown by the plug-in using the function *cciThrowException()* and the tree is populated with data provided in the function parameters.

Output File

The Output file is created as a QSAM data set with record format of fixed (F), or Variable (V), or Fixed Blocked (FB) or Variable Blocked (VB). The file can be open with disposition NEW (for new file) or MOD (to append data to an existing file). The content of the file is the data received in the data messages on the “in” terminal.

File Rename plug-in

Description

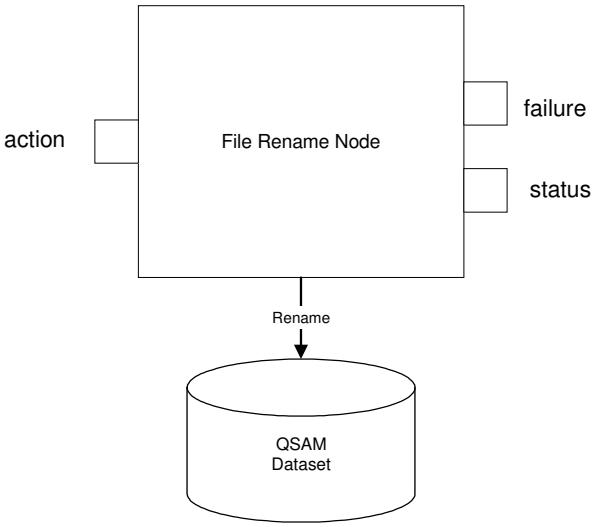

Figure 6 - File Rename

When the File Rename plug-in node receives a rename action message with the old and the new file names, then it changes the file name. A status message is propagated to the “status” terminal to confirm the success of the rename action.

If the new file name, in the action message, is a relative generation data set group (GDG) file name (GDG.STEM(+1)) then the absolute GDG name (GDG.STEM.G00nnV00) is included in the status message as the new file name.

Note: For the rename to a next GDG generation (+1) to work, it is necessary to have a GDG group defined (using IDCAMS) and to have an existing first GDG file created (current GDG generation file (+0) must exist). If there isn’t an existing current GDG generation file, the File Rename raises an error exception.

If an error condition is detected, an exception message is propagated to the “failure” terminal or if the “failure” terminal is not wired then a user exception is thrown to terminate the processing.

Copies of the action and status messages are maintained in the Environment tree of the message flow.

Input

Node Properties

- [O] Default.DebugNode: None, Performance or Full (None is default). The “Performance” option is the same as the “None” option for the FileRename. The “Full” option generates debugging and trace information.
- [O] Default.TraceFilename: string (filename). File to write the debugging and trace information, if TraceFilename is not specified (blank), then the debugging and trace information is written on the broker’s ‘stdout’.

[O] = Optional.

Action terminal

Note: Refer to Appendix A for a description of the elements of the action messages.

Rename File Action Message
The Properties and MQMD elements are required together with the following message in XML format:

```
[M]<FileAdapter>
[M]             <Filename>'string'
[M]                     <Action>RENAME</Action>
[M]                     <Type>OUTPUT</Type>
[M]                     <NewFilename>string</NewFilename>
[M]             </Filename>
[M]</FileAdapter>
```

Note: [M] is a mandatory element.

Local Environment

No data is expected in the Local Environment tree.

Environment

No data is expected in the Environment tree.

Exception List

No data is expected in the Exception List tree.

Output

Note: Refer to Appendix A for a description of the elements of the status and exception messages.

Status Terminal

Opened Input File status message
The Properties and MQMD elements are required together with the following message in XML format:

```
<FileAdapter>
        <Filename>'string'
                <Status>RENAMED</Status>
                <Type>OUTPUT</Type>
                <NewFilename>string</NewFilename>
                <TimeFileRenamed>timestamp</TimeFileRenamed>
        </Filename>
</FileAdapter>
```

Failure Terminal

An exception message is propagated to the "failure" terminal when an error exception is detected by the File Rename plug-in node. The Properties and MQMD elements are required together with the following messages in XML format:

Exception when the action message is invalid

```
<FileAdapter>
        <Exception>FileRename</Exception>
        <ErrorText>string</ErrorText>
        <TimeException>timestamp</TimeException>
</FileAdapter>
```


Exception when the dynalloc function returns errors

```
<FileAdapter>
        <Filename>'string'
                <Exception>FileRename</Exception>
                <Function>dynalloc</Function>
                <ErrorCode>integer</ErrorCode>
                <InformationCode>integer</InformationCode>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```

Exception when the rename function returns errors

```
<FileAdapter>
        <Filename>'string'
                <Exception>FileRename</Exception>
                <Function>rename</Function>
                <NewFilename>'string'</NewFilename>
                <ErrorNumber>integer</ErrorNumber>
                <ErrorText>string</ErrorText>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```

Local Environment

No data is created in the Local Environment tree.

Environment

The Environment tree is used to keep the status of the file process. Copies of action, status and exception messages are added to the environment tree. The following is the tree structure:

Environment.FileAdapter.FileRename: string (Filename)

Environment.FileAdapter.FileRename.ActionMessage
Environment.FileAdapter.FileRename.ActionMessage.Action: string (RENAME)
Environment.FileAdapter.FileRename.ActionMessage.Type: string (OUTPUT)
Environment.FileAdapter.FileRename.ActionMessage.NewFilename: string

Environment.FileAdapter.FileRename.StatusMessage
Environment.FileAdapter.FileRename.StatusMessage.Status: string (RENAMED)
Environment.FileAdapter.FileRename.StatusMessage.Type: string (OUTPUT)
Environment.FileAdapter.FileRename.StatusMessage.TimeFileRenamed: timestamp

Environment.FileAdapter.FileRename.ExceptionMessage
Environment.FileAdapter.FileRename.ExceptionMessage.Exception: String (FileRename)
Environment.FileAdapter.FileRename.ExceptionMessage.Function: String
Environment.FileAdapter.FileRename.ExceptionMessage.NewFilename: String
Environment.FileAdapter.FileRename.ExceptionMessage.ErrorCode: integer
Environment.FileAdapter.FileRename.ExceptionMessage.InformationCode: integer
Environment.FileAdapter.FileRename.ExceptionMessage.ErrorNumber: integer
Environment.FileAdapter.FileRename.ExceptionMessage.ErrorText: string
Environment.FileAdapter.FileRename.ExceptionMessage.TimeException: timestamp

Exception List

The broker creates the Exception List tree when an exception is thrown by the plug-in using the function *cciThrowException()* and the tree is populated with data provided in the function parameters.

File Delete plug-in

Description

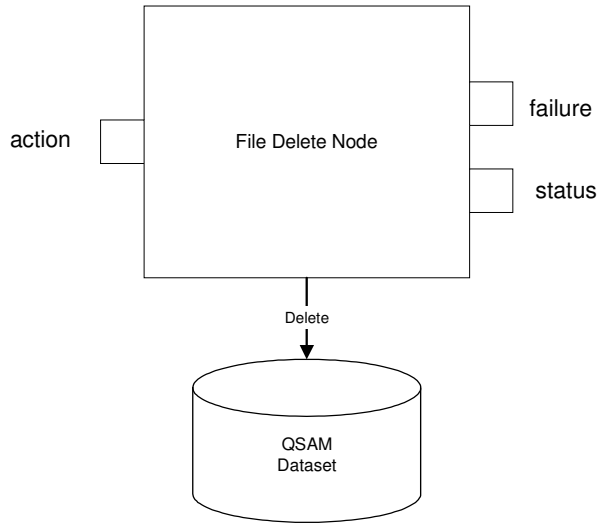

Figure 7 - File Delete

When the File Delete plug-in node receives a delete action message then it removes the file. A status message is propagated to the “status” terminal to confirm the success of the delete action.

If an error condition is detected, an exception message is propagated to the “failure” terminal or if the “failure” terminal is not wired, then a user exception is thrown to terminate the processing.

Copies of the action and status messages are maintained in the Environment tree of the message flow.

Input

Node Properties

- [O] Default.DebugNode: None, Performance or Full (None is default). The “Performance” option is the same as the “None” option for the FileDelete. The “Full” option generates debugging and trace information.
- [O] Default.TraceFilename: string (filename). File to write the debugging and trace information, if TraceFilename is not specified (blank) then the debugging and trace information is written to the broker’s ‘stdout’.

[O] = Optional.

Action terminal

Note: Refer to Appendix A for a description of the elements of the action messages.

Delete File Action Message
The Properties and MQMD elements are required together with the following message in XML format:

```
[M]<FileAdapter>
[M]             <Filename>string
[M]                     <Action>DELETE</Action>
[M]                     <Type>OUTPUT</Type>
```



```
[M]             </Filename>
[M]</FileAdapter>
```

Note: [M] is a mandatory element.

Local Environment

No data is expected in the Local Environment tree.

Environment

No data is expected in the Environment tree

Exception List

No data is expected in the Exception List tree

Output

Note: Refer to Appendix A for a description of the elements of the status and exception messages.

Status Terminal

Opened Input File status message
The Properties and MQMD elements are required together with the following message in XML format:

```
<FileAdapter>
        <Filename>string
                <Status>DELETED</Status>
                <Type>OUTPUT</Type>
                <TimeFileDeleted>timestamp</TimeFileDeleted>
        </Filename>
</FileAdapter>
```

Failure Terminal

An exception message is propagated to the “failure” terminal when an error exception is detected by the File Delete plug-in node. The Properties and MQMD elements are required together with the following messages in XML format:

Exception when the action message is invalid

```
<FileAdapter>
        <Exception>FileDelete</Exception>
        <ErrorText>string</ErrorText>
        <TimeException>timestamp</TimeException>
</FileAdapter>
```

Exception when the remove function returns errors

```
<FileAdapter>
        <Filename>‘string’
                <Exception>FileDelete</Exception>
                <Function>remove</Function>
                <ErrorNumber>integer</ErrorNumber>
                <ErrorText>string</ErrorText>
                <TimeException>timestamp</TimeException>
        </Filename>
</FileAdapter>
```


Local Environment

No data is created in the Local Environment tree.

Environment

The Environment tree is used to keep the status of the file process. Copies of action, status and exception messages are added to the environment tree. The following is the tree structure:

Environment.FileAdapter.FileDelete: string (Filename)

Environment.FileAdapter.FileDelete.ActionMessage
Environment.FileAdapter.FileDelete.ActionMessage.Action: string (DELETE)
Environment.FileAdapter.FileDelete.ActionMessage.Type: string (OUTPUT)

Environment.FileAdapter.FileDelete.StatusMessage
Environment.FileAdapter.FileDelete.StatusMessage.Status: string (DELETED)
Environment.FileAdapter.FileDelete.ActionMessage.Type: string (OUTPUT)
Environment.FileAdapter.FileDelete.StatusMessage.TimeFileDeleted: timestamp

Environment.FileAdapter.FileDelete.ExceptionMessage
Environment.FileAdapter.FileDelete.ExceptionMessage.Exception: String (FileDelete)
Environment.FileAdapter.FileDelete.ExceptionMessage.Function: String
Environment.FileAdapter.FileDelete.ExceptionMessage.ErrorNumber: integer
Environment.FileAdapter.FileDelete.ExceptionMessage.ErrorText: string
Environment.FileAdapter.FileDelete.ExceptionMessage.TimeException: timestamp

Exception List

The broker creates the Exception List tree when an exception is thrown by the plug-in using the function *cciThrowException()* and the tree is populated with data provided in the function parameters.

Chapter 5. Examples using the plug-in nodes

The exported Eclipse project *c:\FileAdapter\FileAdapterTestFlows* for WMB have the Unit Test message flows and the file *c:\FileAdapter\FileAdapterTest.tar.z* has the tools to drive the unit tests.

Unit Test message flows

To test each plug-in node a unit test message flow is used. The following are the characteristics of the four message flows:

1. **File Read message flow,** the input terminal (“action”) is connected to an MQInput node and the output terminals (“status”, “out” and “failure”) are connected to MQOutput nodes. The queue ACTION_QUEUE is wired to the “action” terminal. Action control messages are put on this queue to open input files. The OUT_QUEUE is associated to the “out” terminal. One message per record on the input file is put on the OUT_QUEUE. The RCD node (BLOB to BLOB) is required to force the message flow to make a copy of the message before propagating it to the MQOutput node. If the RCD node is not wired then the File Read node generates an error exception (“Invalid state for finalize”). The STATUS_QUEUE is wired to the “status” terminal. The opened and closed status messages are put on this queue. The FAILURE_QUEUE is wired to the “failure” terminal. The exception messages generated by the plug-in node are put on this queue.

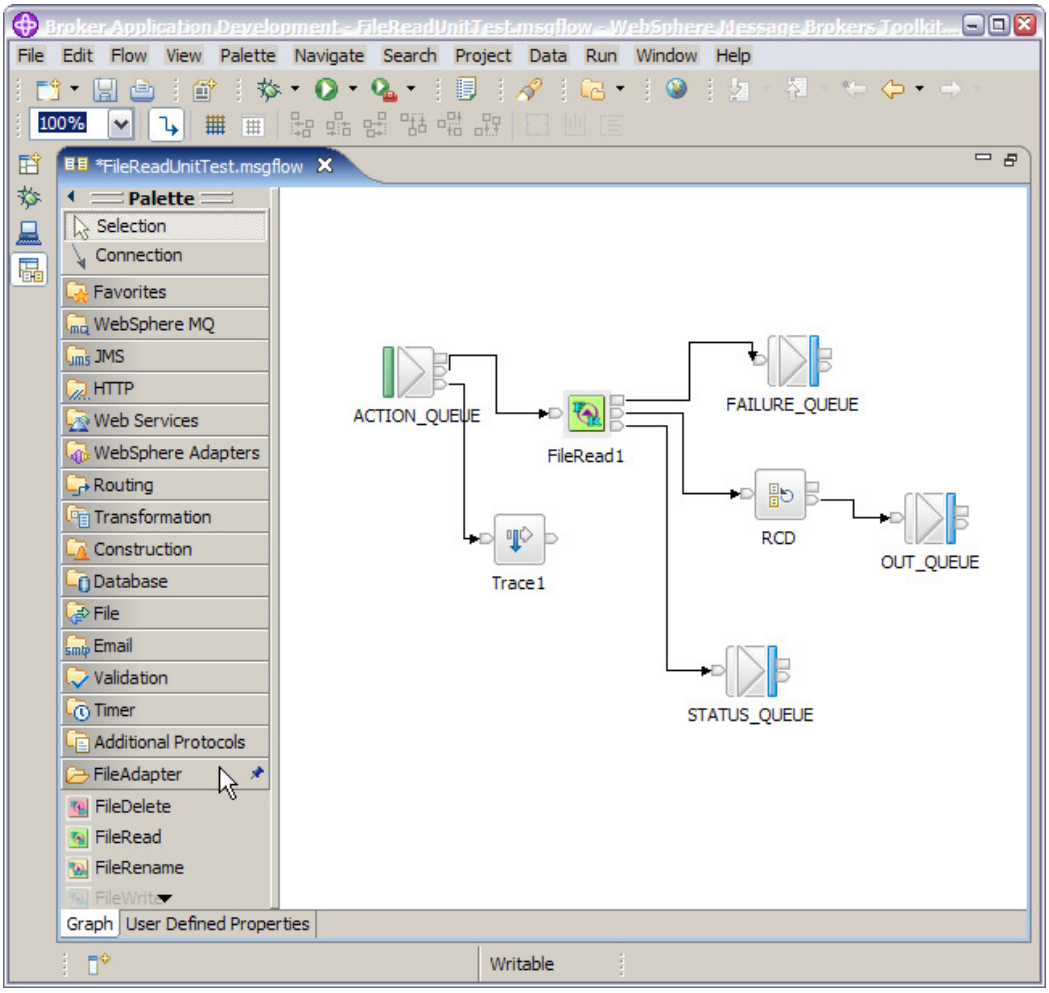

Figure 8 - FileRead unit test flow

2. **File Write message flow,** the File Write node input terminals (“action” and “in”) are connected to a Filter node and the output terminals (“failure” and “status”) are connected to MQOutput nodes. The queue IN_QUEUE is wired to the “in” terminal of the Filter node. Action control messages are put on this queue to open and close output files. One data message per record on the output file is put on the IN_QUEUE. The filter node separates the action messages from the data messages and propagates them to the correct input terminals on the File Write node. The STATUS_QUEUE is wired to the “status” terminal. The “opened” and “closed” status messages are put on this queue. The FAILURE_QUEUE is wired to the “failure” terminal. The exception messages generated by the File Write plug-in node are put on this queue.

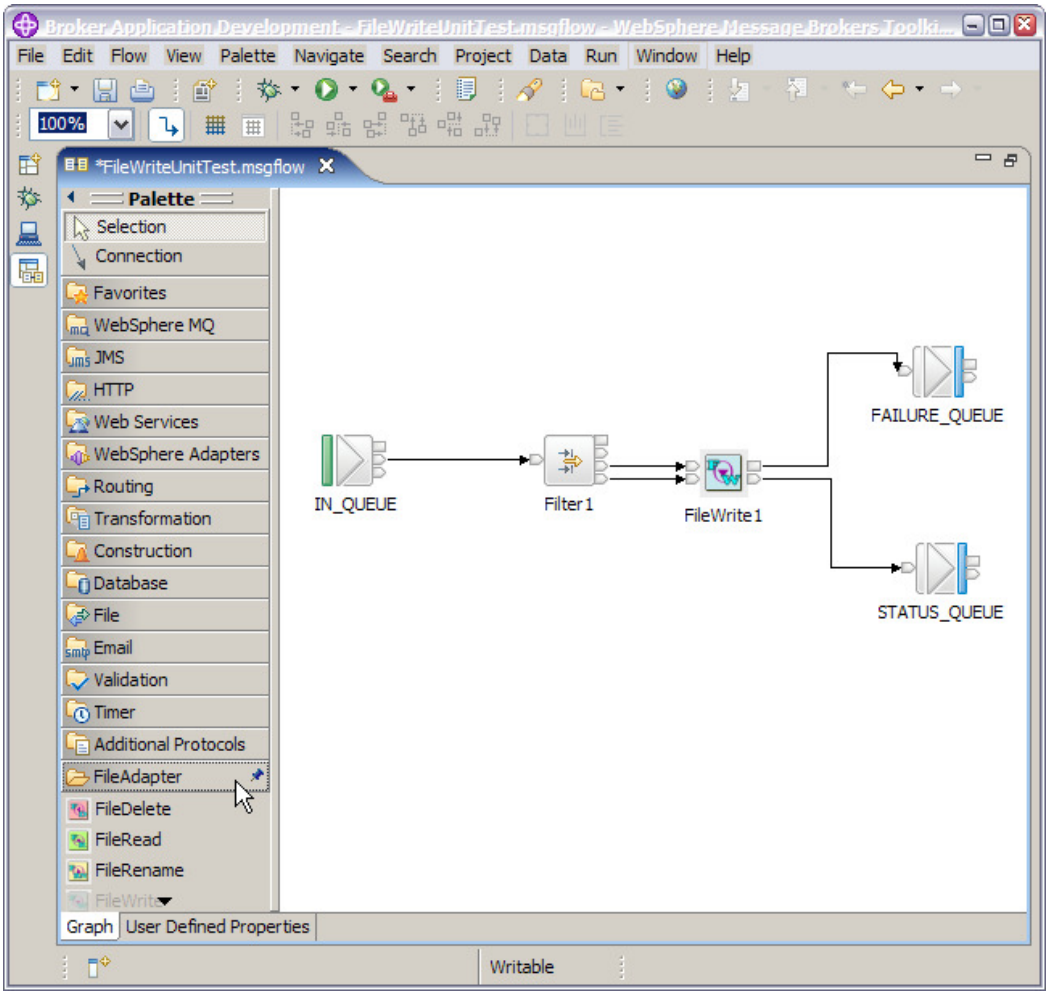

Figure 9 - FileWrite unit test flow

3. **File Rename message flow,** the input terminal (“action”) is connected to an MQInput node and the output terminals (“status” and “failure”) are connected to MQOutput nodes. The queue ACTION_QUEUE is wired to the “action” terminal. Action control messages are put on this queue to rename files. The STATUS_QUEUE is wired to the “status” terminal. The “renamed status message” is put on this queue. The FAILURE_QUEUE is wired to the “failure” terminal. The exception messages generated by the File Rename plug-in node are put on this queue.

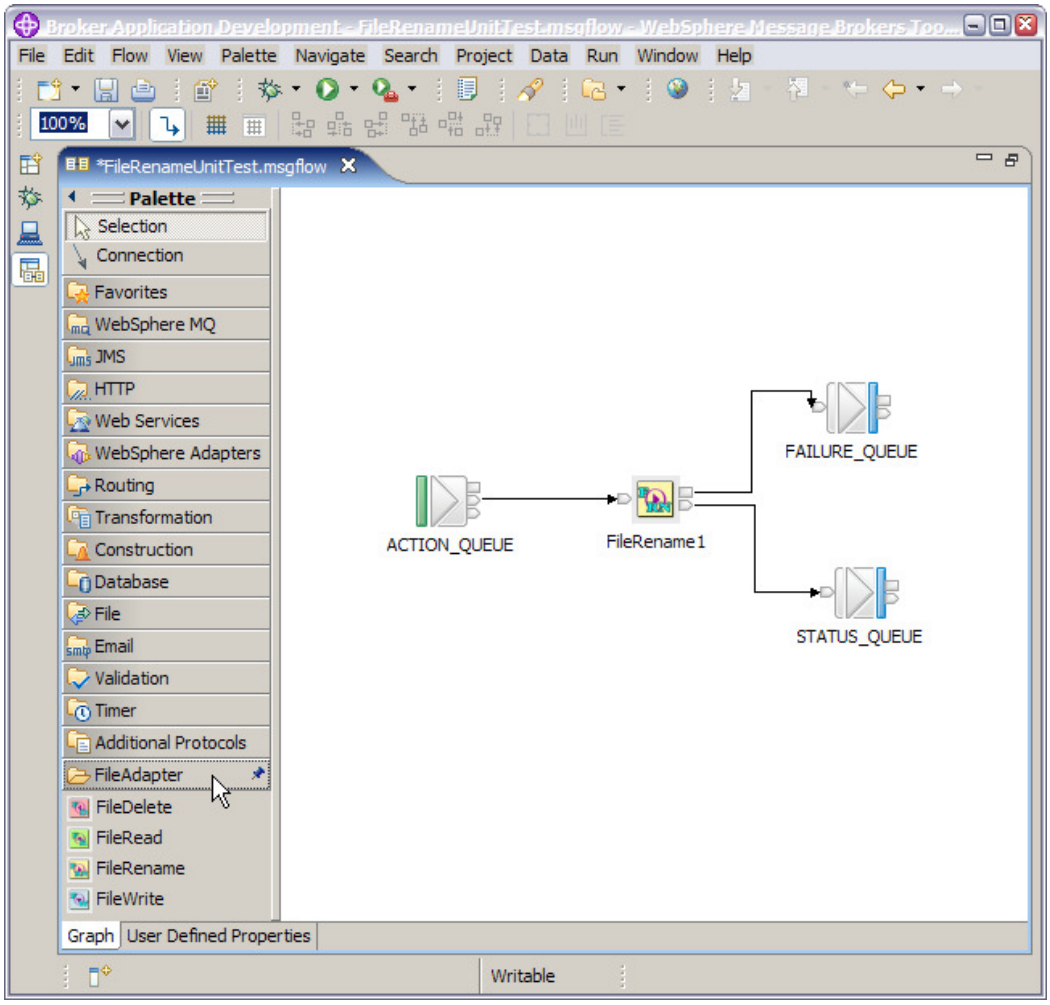

Figure 10 - FileRename unit test flow

4. **File Delete message flow,** the input terminal (“action”) is connected to an MQInput node and the output terminals (“status” and “failure”) are connected to MQOutput nodes. The queue ACTION_QUEUE is wired to the “action” terminal. Action control messages are put on this queue to delete files. The STATUS_QUEUE is wired to the “status” terminal. The “deleted status message” is put on this queue. The FAILURE_QUEUE is wired to the “failure” terminal. The exception messages generated by the File Delete plug-in node are put on this queue.

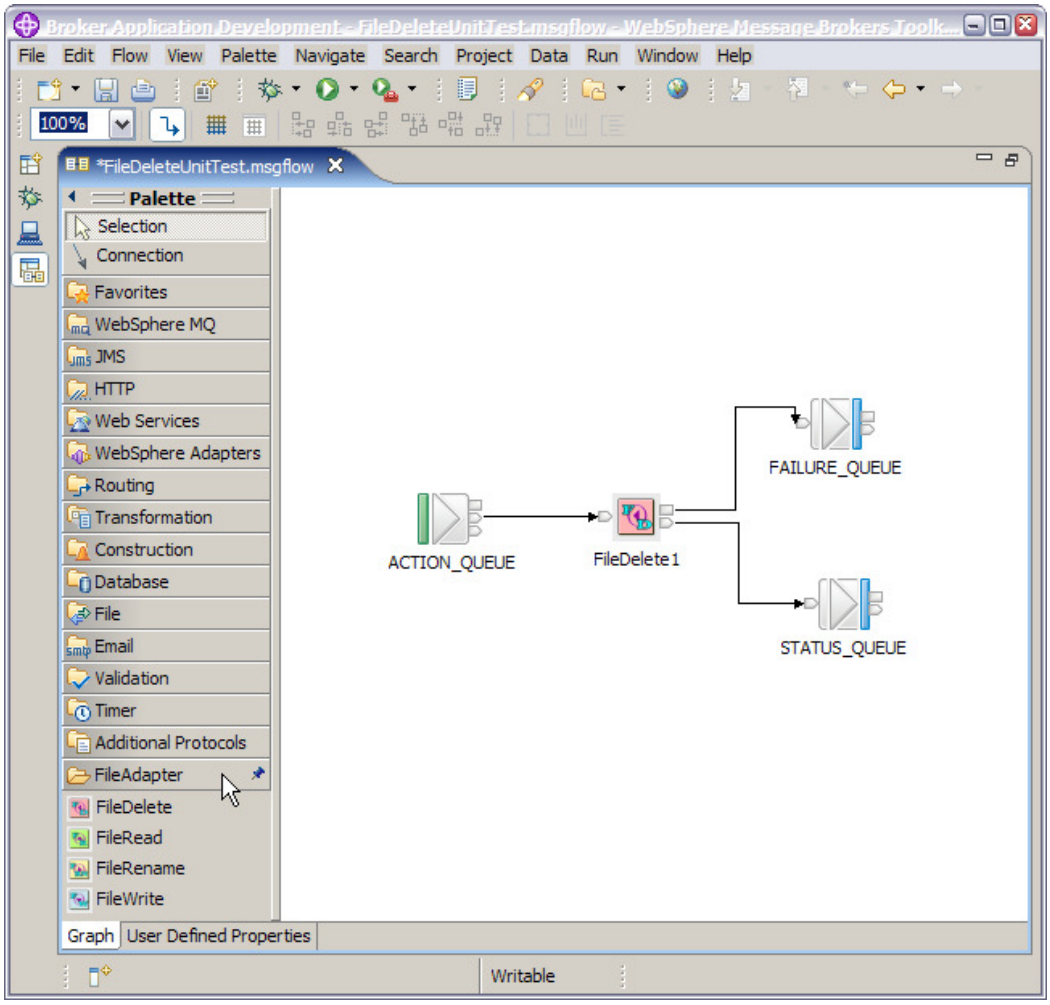

Figure 11 - FileDelete unit test flow

Unit test cases

The Table 1 on page 36 has the Unit Test cases that the test tools provided (in the file “FileAdapterTest.tar.z) can execute.

To execute the unit test cases, it is necessary to do the following steps:

- Install the test tools on USS.
- Create test input files.
- Tailor and create test action messages.
- Define queues for the test message flows.
- Import the test flows into the WMB eclipse toolkit.
- Run the tests:
    - Select the test to run.
    - Select the test message flow.

- Set or review the File Adapter plug-in node attributes.
- Deploy the test message flow to the execution group in the broker.
- Run the test using the “UnitTest.rex” script.
- Verify the results.

Test #	Message Flow	Test Case
U1	**FileReadUnitTest**	Input File (FB) Put open action message on “action” terminal. Read, propagate and close file with 10K records of 1000 bytes. Check data messages on “out” terminal Check status message on “status” terminal
U2	**FileReadUnitTest**	Put invalid action messages on the “action” terminal. Cause exceptions. Check exception messages on “failure” terminal
U3	**FileWriteUnitTest** Non-SMS	Output File (FB) Put Open file action message on “action” terminal using non-SMS file attributes. Put 10 data messages on the “in” terminal. Put Close file action message on “action” terminal. Check status messages on “status” terminal. Check output file contents. Repeat for variable length file (VB).
U4	**FileWriteUnitTest** SMS	Output File (FB) Put Open file action message on “action” terminal using SMS file attributes (Data Class). Put 10 data messages on the “in” terminal. Put Close file action message on “action” terminal. Check status messages on “status” terminal. Check output file contents. Repeat for variable length file (VB).
U5	**FileWriteUnitTest**	Put data messages when output file is not open. Put invalid action messages to the *“action”*

Test #	Message Flow	Test Case
		terminal Cause exceptions Check exception messages on "failure" *terminal*
U6	**FileWriteUnitTest**	Output File (FB) Put open file message on "action" terminal Put 10 messages to the "in" *terminal* Put exception close file message on "action" terminal Check exception message on "failure" *terminal* Check contents of the file (10 records)
U7	**FileWriteUnitTest** FileWrite node in automated operation mode Close file after 5 records	Output File (FB) Put 10 messages to the "in" *terminal* Check status message on "status" *terminal* Check contents of the 2 files (5 records each)
U8	**FileWriteUnitTest** FileWrite node in automated operation mode Close file after 1 minute	Output File (FB) Put 10 messages to the "in" *terminal* Wait 1 minute Put 10 messages to the "in" *terminal* Check status message on "status" *terminal* Check contents of the 2 files (10 records each)

Test #	Message Flow	Test Case
U9	**FileWriteUnitTest** FileWrite node in automated operation mode	Output File (FB) Configure FileWrite node properties for automated mode Put 10 messages to the “in” terminal Put exception close file message on “action” terminal Check exception message on “failure” terminal Check contents of the file (10 records)
U10	**FileWriteUnitTest** FileWrite node in automated operation mode	Output File (FB) Configure an invalid file in the properties. The file cannot be opened. Put 1 message to the “in” terminal Check exception message on “failure” terminal
U11	**FileRenameUnitTest**	Put rename action message to the “action” terminal. Check status message on “status” terminal. Check file with new name. Repeat test to rename to a GDG file.
U12	**FileRenameUnitTest**	Put invalid action messages to the “action” terminal to cause 'unable to rename file' exceptions. Check exception messages on “failure” terminal.
U13	**FileDeleteUnitTest**	Put delete action message to the *“action” terminal*. Check status message on “status” terminal. Check deletion of the file.
U14	**FileDeleteUnitTest**	Put invalid action messages to the “action” terminal to cause 'unable to delete file' exceptions. Check exception messages on “failure” terminal.

Table 1 - Unit Test cases

Test Setup

This section describes how test the four plugin nodes:

- FileRead
- FileWrite
- FileRename
- FileDelete

The following instructions assume that:

- All the files for testing go in **/u/wi01brk/FileAdapterTest**

Test set up on z/OS

Install the test distribution file

On Windows use FTP to copy the test distribution file to the HFS:

```
C:\FileAdapter>ftp HURMVS99
Connected to HURMVS99.hursley.ibm.com.
220-FTPD1 IBM FTP CS V2R10 at HURMVS99.HURSLEY.IBM.COM, 13:31:05 on
2003-02-21.
220 Connection will close if idle for more than 5 minutes.
User (HURMVS99.hursley.ibm.com:(none)): wi01brk
331 Send password please.
Password:
230 WI01BRK is logged on.  Working directory is "WI01BRK.".
ftp> cd /u/wi01brk
250 HFS directory /u/wi01brk is the current working directory
ftp> bin
200 Representation type is Image
ftp> put FileAdapterTest.tar.z
200 Port request OK.
125 Storing data set /u/wi01brk/FileAdapter.tar.z
250 Transfer completed successfully.
ftp: 102400 bytes sent in 0.47Seconds 217.87Kbytes/sec.
ftp> bye
221 Quit command received. Goodbye.

C:\FileAdapter>
```

On z/OS (USS) **decompress** the test distribution file:

```
/u/wi01brk:>tar -xvozf FileAdapterTest.tar.z
x FileAdapterTest, 0 bytes, 0 tape blocks
x FileAdapterTest/ccl, 391 bytes, 1 tape block
x FileAdapterTest/createTestFile, 94208 bytes, 184 tape blocks
x FileAdapterTest/createTestFile.c, 5282 bytes, 11 tape blocks
x FileAdapterTest/createTestFile.o, 8320 bytes, 17 tape blocks
x FileAdapterTest/filereadopenF10.dat, 181 bytes, 1 tape block
x FileAdapterTest/filereadopenF10.txt, 217 bytes, 1 tape block
x FileAdapterTest/filereadopenF10.xml, 249 bytes, 1 tape block
x FileAdapterTest/filereadopenF100.dat, 183 bytes, 1 tape block
x FileAdapterTest/filereadopenF100.xml, 253 bytes, 1 tape block
x FileAdapterTest/filereadopenF1000.xml, 253 bytes, 1 tape block
x FileAdapterTest/filereadopenV10.xml, 233 bytes, 1 tape block
x FileAdapterTest/filereadopenV100.xml, 252 bytes, 1 tape block
x FileAdapterTest/filereadopenV1000.xml, 253 bytes, 1 tape block
x FileAdapterTest/filereadInvalid0.dat, 180 bytes, 1 tape block
x FileAdapterTest/filereadInvalid0.xml, 250 bytes, 1 tape block
x FileAdapterTest/filereadInvalid1.dat, 181 bytes, 1 tape block
x FileAdapterTest/filereadInvalid1.xml, 251 bytes, 1 tape block
x FileAdapterTest/filereadInvalid2.dat, 181 bytes, 1 tape block
```



```
x FileAdapterTest/filereadInvalid2.xml, 251 bytes, 1 tape block
x FileAdapterTest/filereadInvalid3.dat, 31 bytes, 1 tape block
x FileAdapterTest/filereadInvalid3.xml, 34 bytes, 1 tape block
x FileAdapterTest/oneline.rex, 828 bytes, 2 tape blocks
x FileAdapterTest/MQgetc, 12288 bytes, 24 tape blocks
x FileAdapterTest/MQgetf, 90112 bytes, 176 tape blocks
x FileAdapterTest/MQgetf.c, 15989 bytes, 32 tape blocks
x FileAdapterTest/MQgetf.o, 8080 bytes, 16 tape blocks
x FileAdapterTest/MQputf, 90112 bytes, 176 tape blocks
x FileAdapterTest/MQputf.c, 14170 bytes, 28 tape blocks
x FileAdapterTest/MQputf.o, 8320 bytes, 17 tape blocks
x FileAdapterTest/U1.deadq, 0 bytes, 0 tape blocks
x FileAdapterTest/U1.failure, 0 bytes, 0 tape blocks
x FileAdapterTest/U1.out, 0 bytes, 0 tape blocks
x FileAdapterTest/U1.rex, 618 bytes, 2 tape blocks
x FileAdapterTest/U1.status, 0 bytes, 0 tape blocks
x FileAdapterTest/U2.deadq, 0 bytes, 0 tape blocks
x FileAdapterTest/U2.failure, 238 bytes, 1 tape block
x FileAdapterTest/U2.out, 0 bytes, 0 tape blocks
x FileAdapterTest/U2.rex, 746 bytes, 2 tape blocks
x FileAdapterTest/U2.status, 0 bytes, 0 tape blocks
/u/wi01brk:>
```

Update your STEPLIB environment variable

The supplied tests use two programs, MQputf and MQgetf. To run these programs your STEPLIB environment variable must point to the WebSphere MQ runtime libraries:

On z/OS (USS) edit your profile:

```
/u/wi01brk:>oedit .profile
```

Insert the following statements *after* other “export STEPLIB” statements, if any:

```
# STEPLIB needed to run MQ programs
export STEPLIB=£STEPLIB:"MQM.V600.SCSQAUTH:MQM.V600.SCSQANLE"
```

Save the update profile and run it:

```
/u/wi01brk:>. .profile
/u/wi01brk:>
```

If you are not running WebSphere MQ 6.0

The supplied MQputf and MQgetf programs run with WebSphere MQ 6.0. If you have a different release, compile and link MQputf.c and MQgetf.c using the appropriate WebSphere MQ libraries. For example, to compile MQputf.c:

```
c89 -g -c -Wc,SS,long,langlvl\(extended\) -I"//'MQM.V600.SCSQC370'"
-o MQputf.o MQputf.c
```

Replace **MQM.V600.SCSQC370** by the appropriate SCSQC370 library.

The shell script “ccl” can be used instead of entering the above command. For example, enter **ccl MQputf** to compile MQputf.c.

To link:

```
cc -o MQputf MQputf.o "//'MQM.V600.SCSQLOAD(CSQBSTUB)'"
```

Replace **MQM.V600.SCSQLOAD** by the appropriate SCSQLOAD library.

Input Files

The tests use four different input files:

1. A fixed record length file ('F'), where the record format is 2 byte packed decimal number, 4 byte number, 2 byte number, 4 byte length of the character string and a 988 byte character string of mixed case. The total length of each record is 1,000 bytes.
2. A fixed record length blocked file ('FB'), where the record format will be the same as above, but using a block size of a multiple of the record length.
3. A variable record length file ('V'), where the record format is 2 byte packed decimal number, 4 byte number, 2 byte number, 4 byte length of the character string and a maximum of 988 byte character string of mixed case plus 4 bytes for the RDW (record descriptor word). The variable record length test file will contain records of the two 4 byte numbers followed by a mixture of 10 byte, 90 byte and 988 byte character strings of upper case characters.
4. A variable record length blocked file ('VB'), where the record format will be the same as above, but using a block size of a multiple of the maximum record length plus 4 bytes for the BDW (block descriptor word).

Create the files for testing

On z/OS (USS), run the following REXX program to create the z/OS test data sets:

```
/u/wi01brk/FileAdapterTest:>createTestFiles.rex
createTestFile start
Values received: Number of records = 0010, Record format = F
DDname is SYS00004
Data set name is WI01BRK.FILEREAD.F10
Allocate success!
Output Deallocate success!
createTestFile end
…
createTestFile start
Values received: Number of records = 10000, Record format = V
DDname is SYS00004
Data set name is WI01BRK.FILEREAD.V10000
Allocate success!
Output Deallocate success!
createTestFile end
IGD01008I THIS DATA SET WILL BE SMS MANAGED
```

Note: You might not have authority to execute the following TSO commands in your system then you will have to run a batch job to create a test GDG group and the first GDG file (this is required by the FileRename unit tests).

```
def gdg(name('WI01BRK.GDG') noempty scratch limit(5))
IGD01008I THIS DATA SET WILL BE SMS MANAGED
alloc da('WI01BRK.GDG.G0001V00') new catalog
/u/wi01brk/FileAdapterTest:>
```

The above creates these data sets (as displayed by (TSO) ISPF option 3.4):

```
DSLIST - Data Sets Matching WI01BRK.FILEREAD
Command ===>

Command - Enter "/" to select action
-------------------------------------------
         WI01BRK.FILEREAD.F10
         WI01BRK.FILEREAD.F100
         WI01BRK.FILEREAD.F1000
         WI01BRK.FILEREAD.F10000
         WI01BRK.FILEREAD.V10
         WI01BRK.FILEREAD.V100
         WI01BRK.FILEREAD.V1000
         WI01BRK.FILEREAD.V10000
```



```
DSLIST - Data Sets Matching WI01BRK.GDG
Command ===>

Command - Enter "/" to select action
-------------------------------------
         WI01BRK.GDG
         WI01BRK.GDG.G0001V00
```

Tailor the Action Message files

On z/OS (USS), run the following REXX program to create the Control Message files:

```
/u/wi01brk/FileAdapterTest:>tailorActionMsgs.rex
```

The REXX program edits all the files that need tailoring. Follow the instructions in the first line of the file. For example:

```
EDIT       filewriteOpenFs002.xml
Command ===>
****** ***************************************************************
Top of Data
000001 <!-- Change DataClass to one that is valid in your
installation -->
000002 <FileAdapter>
000003    <Filename>%USERID%.FILEWRTE.F10.S002
000004        <Action>OPEN</Action>
000005        <Type>OUTPUT</Type>
000006        <Attributes>
000007            <Disposition>
000008               <Status>NEW</Status>
000009               <Normal>CATLG</Normal>
000010               <Conditional>DELETE</Conditional>
000011            </Disposition>
000012            <DataClass>VSUAREZF</DataClass>
000013        </Attributes>
000014    </Filename>
000015 </FileAdapter>
```

Create the Action Message files

On z/OS (USS), run the following REXX program to create the Control Message files:

```
/u/wi01brk/FileAdapterTest:>createActionMsgFiles.rex
Converting filereadInvalid1.xml to filereadInvalid1.dat
Converting filereadInvalid2.xml to filereadInvalid2.dat
Converting filereadInvalid3.xml to filereadInvalid3.dat
Converting filereadopenF10.xml to filereadopenF10.dat
Converting filereadopenF100.xml to filereadopenF100.dat
Converting filereadopenF1000.xml to filereadopenF1000.dat
Converting filereadopenV10.xml to filereadopenV10.dat
Converting filereadopenV100.xml to filereadopenV100.dat
Converting filereadopenV1000.xml to filereadopenV1000.dat
/u/wi01brk/FileAdapterTest:>
```

The program above converts the supplied skeleton XML Control Message files to input messages you will use to run the tests. For example, the following XML file (`filereadopenF10.xml`):

```
<FileAdapter>
   <Filename>'%USERID%.TESTFILE.F10'
      <Action>OPEN</Action>
      <Type>INPUT</Type>
      <Attributes>
```



```
        <Disposition>
           <Status>OLD</Status>
        </Disposition>
      </Attributes>
   </Filename>
</FileAdapter>
```

Becomes the one-line file `filereadopenF10.dat`, which is suitable for input to the MQputf program:

```
/u/wi01brk/FileAdapterTest:>cat filereadopenF10.dat
<FileAdapter><Filename>'WI01BRK.TESTFILE.F10'<Action>OPEN</Action><T
ype>INPUT</Type><Attributes><Disposition><Status>OLD</Status></Dispo
sition></Attributes></Filename></FileAdapter>(146)WI01BRK:/u/wi01brk
/FileAdapterTest:>
```

Note that the program replaced the token **%USERID%** by the current userid.

Create the queues for testing

Using (batch job) CSQUTIL, the (TSO) ISPF panels or (from Windows) “runmqsc –x”, create the queues for testing:

```
define qlocal(ACTION) replace
define qlocal(OUT) replace
define qlocal(FAILURE) replace
define qlocal(STATUS) replace
define qlocal(EXCEPTION) replace
```

This completes the Test setup on z/OS.

Unit Test set up on Windows

This section describes how to import the test messages flows. The import process is different for the two supported versions of WebSphere Business Integration Message Broker and they are described in:

- “Installation of the test message flows on WMQI Control Center” on page 41.
- “Installation of the test message flows on the WMB toolkit” on page 42.

Installation of the test message flows on WMQI Control Center

Import the example flows

From the Control Center Message Flows tab, import the message flows in **c:\FileAdapter\examples**:

File → Import to Workspace …

Click on **Browse …**; go to the **C:\FileAdapter\examples** directory; select the files:

```
c:\FileAdapter\examples\FileDeleteUnitTest.xml
c:\FileAdapter\examples\FileReadUnitTest.xml
c:\FileAdapter\examples\FileRenameUnitTest.xml
c:\FileAdapter\examples\FileWriteUnitTest.xml
```

Click on **Import**.

Check in the imported message flows.

Installation of the test message flows on the WMB toolkit

- Start the Message Broker Toolkit.
- Open the "Broker Application Development" perspective.
- Click on File -> Import -> Existing project into workspace.
- Click **Next**.
- Click **Browse** on Project Contents.
- Expand the directories until "**C:\FileAdapter\FileAdapterTestFlows**" is found.
- Click **OK.**
- Click **Finish**.
- On the Resource Navigator view, right click on the "FileAdapterTestFlows" project.
- Click on **Rebuild Project**.
- Repeat the same for the project "**C:\FileAdapter\FileAdapterTest_MessageSet".** This is required for the scenario tests.

File Read Unit Test

Assign and deploy the test message flow

Select the message flow **FileReadUnitTest**, review the **FileRead** node attributes, and deploy to the **default** execution group. Verify using the **Log** that the deploy operation succeeded.

Run the File Read unit test U1

A REXX program ("`UnitTest.rex`") implements the unit test cases (U1, U2, etc., as described in *File Adapter Acceptance Test Plan*).

To run a test case, enter:

```
UnitTest.rex <Test Case> <Queue Manager name>
```

Where *Test Case* is the test case number and *Queue Manager Name* is the broker's queue manager.

For example, to run test case U1:

```
/u/wi01brk/FileAdapterTest:>UnitTest.rex U1 WI01
MQputf ACTION WI01 < filereadopenF10.dat
Sample AMQSPUT0 start
target queue is ACTION
Sample AMQSPUT0 end
MQputf ACTION WI01 < filereadopenV10.dat
Sample AMQSPUT0 start
target queue is ACTION
Sample AMQSPUT0 end
Expected results:
20 data messages on the out queue
4 status messages on the status queue (OPENED+CLOSED)
ls -ltr U1.*
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  6 15:56 U1.failure
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  6 15:56 U1.deadq
-rw-r-----   1 WI01BRK  HSTGRP      1330 Mar  6 15:56 U1.status
```



```
-rw-r-----   1 WI01BRK  HSTGRP     14070 Mar  6 15:56 U1.out
/u/wi01brk/FileAdapterTest:>
```

The UnitTest.rex program performs the following actions to run test case U1:

Puts an action message using the command:

```
MQputf ACTION <qmgr name> < filereadopenF10.dat
```

If successful, the message flow puts two status messages on the STATUS queue and ten messages (one per record in the input file ***<userid>*.FILEREAD.F10**) on the OUT queue.

U1.rex waits 3 seconds and then reads all the queues:

```
MQgetf WI01.DEAD_QUEUE <Queue Manager name> NOWAIT > U1.deadq
MQgetf FAILURE <Queue Manager name> NOWAIT > U1.failure
MQgetf STATUS <Queue Manager name> NOWAIT > U1.status
MQgetf OUT <Queue Manager name> NOWAIT > U1.out
```

Finally, it lists the files resulting from reading the queues. If the test runs successfully, U1.deadq and U1.failure should be empty:

```
ls -ltr U1.*
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  6 15:56 U1.failure
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  6 15:56 U1.deadq
-rw-r-----   1 WI01BRK  HSTGRP      1330 Mar  6 15:56 U1.status
-rw-r-----   1 WI01BRK  HSTGRP     14070 Mar  6 15:56 U1.out
```

The file U1.status contains:

```
/u/wi01brk/FileAdapterTest:>cat U1.status
PutTime: 16293471     message
<<FileAdapter><Filename>&apos;WI01BRK.FILEREAD.F10&apos;<Status>OPEN
ED</Status><Type>INPUT</Type><Attributes><Disposition><Status>OLD</S
tatus></Disposition></Attributes><TimeFileOpened>Mon Feb 24 16:29:14
2003</TimeFileOpened></Filename></FileAdapter>>

PutTime: 16293471     message
<<FileAdapter><Filename>&apos;WI01BRK.FILEREAD.F10&apos;<Status>CLOS
ED</Status><Type>INPUT</Type><Attributes><Disposition><Status>OLD</S
tatus></Disposition></Attributes><TimeFileOpened>Mon Feb 24 16:29:16
2003</TimeFileOpened><TimeFileClosed>Mon Feb 24 16:29:16
2003</TimeFileClosed><RecordsProcessed>10</RecordsProcessed></Filena
me></FileAdapter>>
/u/wi01brk/FileAdapterTest:>
```

Run the File Read unit test U2

To run test case U2, enter:

```
/u/wi01brk/FileAdapterTest:>UnitTest.rex U2 WI01
```

The program puts incorrect action messages to perform the test case U2.

File Write Unit Test

Assign and deploy the test message flow

Select the message flow **FileWriteUnitTest**, review the **FileWrite** node attributes, and deploy to the **default** execution group. Verify using the **Log** that the deploy operation succeeded.

Run the File Write unit tests U3 and U4

Cases U3 to U4 test the following functions:

- File open
  - With non-SMS attributes
  - With SMS attributes
- File write
- File close

To run test case U3, use the REXX program `UnitTest.rex`:

`UnitTest.rex U3 <Queue Manager name>`

Where *Queue Manager Name* is the broker's queue manager.

```
/u/wi01brk/FileAdapterTest:>UnitTest.rex U3 WI01
MQputf IN WI01 < filewriteOpenFs001.dat
Sample AMQSPUT0 start
target queue is IN
Sample AMQSPUT0 end
MQputf IN WI01 < filewritedata.dat
Sample AMQSPUT0 start
target queue is IN
Sample AMQSPUT0 end
…
MQputf IN WI01 < filewriteCloseFs001.dat
Sample AMQSPUT0 start
target queue is IN
Sample AMQSPUT0 end
Expected results:
2 status messages on the status queue (OPENED/CLOSED)
1 output file on TSO called WI01BKR.FILEWRTE.F10.S001
To re-run this test, delete data set WI01BRK.FILEWRTE.F10.S001
ls -ltr U3.*
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  7 09:16 U3.failure
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  7 09:16 U3.deadq
-rw-r-----   1 WI01BRK  HSTGRP      1133 Mar  7 09:16 U3.status
-rw-r-----   1 WI01BRK  HSTGRP         0 Mar  7 09:16 U3.out
/u/wi01brk/FileAdapterTest:>
```

Use **oedit** to check the contents of U3.status.

Use **ISPF option 3.4**, to check that the file exists

```
DSLIST - Data Sets Matching WI01BRK.FILEWRTE
Command ===>

Command - Enter "/" to select action
-------------------------------------------
         WI01BRK.FILEWRTE.F10.S001
```

To run test case U4, use the REXX program `UnitTest.rex`:

`UnitTest.rex U4 <Queue Manager name>`

Where *Queue Manager Name* is the broker's queue manager.

```
/u/wi01brk/FileAdapterTest:>UnitTest.rex U4 WI01
MQputf IN WI01 < filewriteOpenFs002.dat
Sample AMQSPUT0 start
target queue is IN
Sample AMQSPUT0 end
MQputf IN WI01 < filewritedata.dat
```



```
Sample AMQSPUT0 start
target queue is IN
Sample AMQSPUT0 end
…
MQputf IN WI01 < filewriteCloseFs002.dat
Sample AMQSPUT0 start
target queue is IN
Sample AMQSPUT0 end
Expected results:
2 status messages on the status queue (OPENED/CLOSED)
1 output file on TSO called WI01BKR.FILEWRTE.F10.S002
```

Use **oedit** to check the contents of U4.status.

Use **ISPF option 3.4**, to check that the file exists

```
DSLIST - Data Sets Matching WI01BRK.FILEWRTE
Command ===>

Command - Enter "/" to select action
-----------------------------------------
         WI01BRK.FILEWRTE.F10.S002
```

Run the File Write unit test U5

Case U5 tests the following functions:

- It puts a series of invalid action messages to test the exception handling of the File Write plug-in node.
- Exception messages are propagated to the “failure” terminal

Run the File Write unit test U6

Case U6 tests the following function:

- It puts an Exception Close action message to force file close and to generate an exception message.
- Exception messages are propagated to the “failure” terminal

Run the File Write unit tests in Automated Mode

To run in automated mode, check the File Write node attributes Automated Mode and Multithread File on the default tab.

For unit tests U7 – U10, it is necessary to deploy the FileWriteUnitTest message flow with the following FileWrite node attributes on the Automated tab.

Attribute	U7	U8	U9	U10
Maximum Number of Records	5			
Maximum Open Time (min)		1	1	1
Filename	WI01BRK.AUTO	WI01BRK.AUTO	WI01BRK.AUTO	INVALIDFILENAME

Run the File Write unit test U7

Case U7 tests automated mode. To run this test case you must modify the File Write message flow.

- **Check out (or edit)** the flow **FileWriteUnitTest**.
- **Right-click** on the **FileWrit**e node, select **Properties**.
- Select the **Automated Mode** tab.
- Enter **5** in the **Maximum Number of Records** field.
- **Check in (or save)** the flow and **deploy**.

On OMVS, run the U7 test case:

UnitTest.rex U7 ***<Queue Manager name>***

Run the File Write unit test U8

Case U8 tests automated mode. To run this test case you must modify the File Write message flow.

- **Check out (or edit)** the flow **FileWriteUnitTest**.
- **Right-click** on the **FileWrit**e node, select **Properties**.
- Select the **Automated Mode** tab.
- Enter **0** in the **Maximum Number of Records** field.
- Enter **1** in the **Maximum Open Time (minutes)** field.
- **Check in (or save)** the flow and **deploy**.

On OMVS, run the U8 test case:

UnitTest.rex U8 ***<Queue Manager name>***

Run the File Write unit test U9

Case U9 tests automated mode. U9 tests EXCEPTION CLOSE action message.
This case puts 5 data messages followed by an EXCEPTION CLOSE action message.

On OMVS, run the U9 test case:

UnitTest.rex U9 ***<Queue Manager name>***

Run the File Write unit test U10

Case U10 tests automated mode. U10 tests exceptions when there are invalid node attributes to prevent the file to be opened.

- C**heck out (or edit)** the flow **FileWriteUnitTest**.
- **Right-click** on the **FileWrit**e node, select **Properties**.
- Select the **Automated Mode** tab.

- Enter **INVALIDFILENAME** in the **Filename** field.
- **Check in (or save)** the flow and **deploy**.

On OMVS, run the U10 test case:

UnitTest.rex U10 <Queue Manager name>

File Rename Unit Test

Assign and deploy the test message flow

Select the message flow **FileRenameUnitTest**, review the **FileRename** node attributes, and deploy to the **default** execution group. Verify using the **Log** that the deploy operation succeeded.

Run the File Rename unit test U11

Case U11 tests renaming data sets. It also renames a normal data set to a relative GDG name (+1).

On OMVS, run the U11 test case:

UnitTest.rex U11 <Queue Manager name>

Run the File Rename unit test U12

Case U12 tests exception handling when invalid rename action messages are received.

On OMVS, run the U12 test case:

UnitTest.rex U12 <Queue Manager name>

File Delete Unit Test

Assign and deploy the test message flow

Select the message flow **FileDeleteUnitTest**, review the **FileDelete** node attributes, and deploy to the **default** execution group. Verify using the **Log** that the deploy operation succeeded.

Run the File Delete unit test U13

Case U13 tests deleting data sets.

On OMVS, run the U13 test case:

UnitTest.rex U13 <Queue Manager name>

Run the File Delete unit test U14

Case U14 tests exception handling when invalid delete action messages are received.

On OMVS, run the U14 test case:

UnitTest.rex U14 *<Queue Manager name>*

Scenario Test cases

There are several messages included that are scenario tests.

For example:

To test reading a 10000 record file, propagating each record, parsing each record using MRM, doing a transformation to the data and writing all the transformed records in an output file.

- Modify the ESQL in the “PrepareRename” node to have a valid GDG dataset name.
- Deploy message flow F1F1 and the FileAdapterTest_MessageSet.
- On OMVS, run the following command:
    - **STest.rex S1 <Queue Manager Name>**

Appendix A.

Description of the elements of the action, status and exception messages

Element	Value	Description
Action	OPEN, CLOSE, RENAME, DELETE	Action to perform to a file
Type	INPUT, OUTPUT	Type of file
Status	OPENED, CLOSED, RENAMED, DELETED	Status of a file after an action is performed
Filename	'String'	File Name. It could also be a relative GDG name. The quotes are required around the filenames.
NewFilename	'String'	New filename to be used for the rename action. The quotes are required around the filenames.
Disposition.Status	NEW, OLD, SHR, MOD	NEW and SHR are for input files. OLD and MOD are for output files.
Disposition.Normal	KEEP, CATLG, DELETE, UNCATLG	What happens to the file after is closed and deallocated
Disposition.Conditional	KEEP, CATLG, DELETE, UNCATLG	What happens to the file after an error exception that terminates the broker
RecordFormat	F, V, FB, VB	Fixed, Variable, Fixed Blocked and Variable Blocked
RecordLength	Integer	Number of bytes contained in a record
BlockSize	Integer	Number of bytes contained in a block
AllocationUnit	CYL, TRK	Unit to be used to calculate the space required for a new output file
Primary	Integer	Number of allocation units to be used as intial space allocation for a new file

Element	Value	Description
Secondary	Integer	Number of allocation unit to be used for each additional space allocation for a new file (Maximum 16 additional allocations are allowed)
Unit	String	Type of storage device to be used for a new file
DataClass	String	SMS data class to specify file attributes and space requirements.
ManagementClass	String	SMS Management Class to specify how is the file managed by SMS.
StorageClass	String	SMS Storage Class to specify the type of storage device to be used for this file.
TimeFileOpened	Timestamp	Date and Time when a file was opened.
TimeFileClosed	Timestamp	GMT (UTC) Date and Time when a file was closed.
TimeFileRenamed	Timestamp	GMT (UTC) Date and Time when a file was renamed.
TimeFileDeleted	Timestamp	GMT (UTC) Date and Time when a file was deleted.
RecordsProcessed	Integer	Number of records read or written from/to a file.
RecordsTruncated	Integer	Number of truncated (data received is longer than RecordLength) records written to a file.
Exception	String	Name of the node that detected the exception.
Function	String	Internal C function that cause the exception.
ErrorCode	integer	Return code received from the DYNALLOC function.
InformationCode	integer	Return code received from the DYNALLOC function.
ErrorNumber	integer	Error number returned by the errno() function

Element	Value	Description
ErrorText	String	Text associated with an error exception
TimeException	Timestamp	Time when the exception occurred.

Appendix B.

Dynalloc error and information codes

The FileAdapter uses the DYNALLOC call to allocate and open the datasets.

Exception messages can include dynalloc error and information codes when there are problems opening the QSAM files.

These error and information codes are described in the z/OS MVS: Programming Authorized Assembler Services Guide, SA22-7608.

Dynalloc error example

The FileRead node is trying to open a dataset that does not exist or the name is incorrect and the following exception message is generated on the failure terminal of the FileRead node:

```
<FileAdapter>
       <Filename>&apos;WI02BRK.NOFILE.FILE&apos;
              <Exception>FileRead</Exception>
              <Function>dynalloc</Function>
              <ErrorCode>1708</ErrorCode>
              <InformationCode>0002</InformationCode>
              <TimeException>Mon Feb  4 13:01:25 2008 </TimeException>
       </Filename>
</FileAdapter>
```

What is the meaning of the dynalloc error code 1708 and information code 0002?

Searching the manual z/OS MVS: Programming Authorized Assembler Services Guide, SA22-7608-09 for a section called “Interpreting error reason codes from DYNALLOC”.

Table 26-9. Class 7 Error Reason Codes (System Routine Error) shows error code 17zz with return code zz = 08.

The manual says:

“**Meaning**: One of the following occurred:

- The data set name specified is in error. (program error)
- A system error occurred when processing the data set name.

In the request block, the S99INFO field contains the text unit key instead of an information reason code. See the corresponding message for the specific error.

Application Programmer Action: Ensure that the correct data set name was specified. If a generation data group (GDG) level of index was coded for a non-GDG data set, remove the level of index and resubmit the job. Otherwise, this is probably a system error. Resubmit the request. If the problem persists, report the associated messages and DYNALLOC error codes to your system programmer.

System Programmer Action: If the problem recurs and no installation action corrects the problem, search problem reporting data bases for a fix for the problem. If no fix exists, contact the appropriate IBM support personnel.

Corresponding Messages: IKJ56228I or IKJ56229I”

The meaning of the information code (referred as S99INFO in the manual) depends on the error code. The manual in section “Interpreting Information Reason Codes from DYANLLOC” has a description on how to interpret these codes. In this case the information code 0002 does not have value for the FileAdapter user. It means that the error is related to the second parameter in the dynalloc call and this is only meaningful to the IBM software service personnel (in this case is the dataset name).

---------- End of Document ----------

