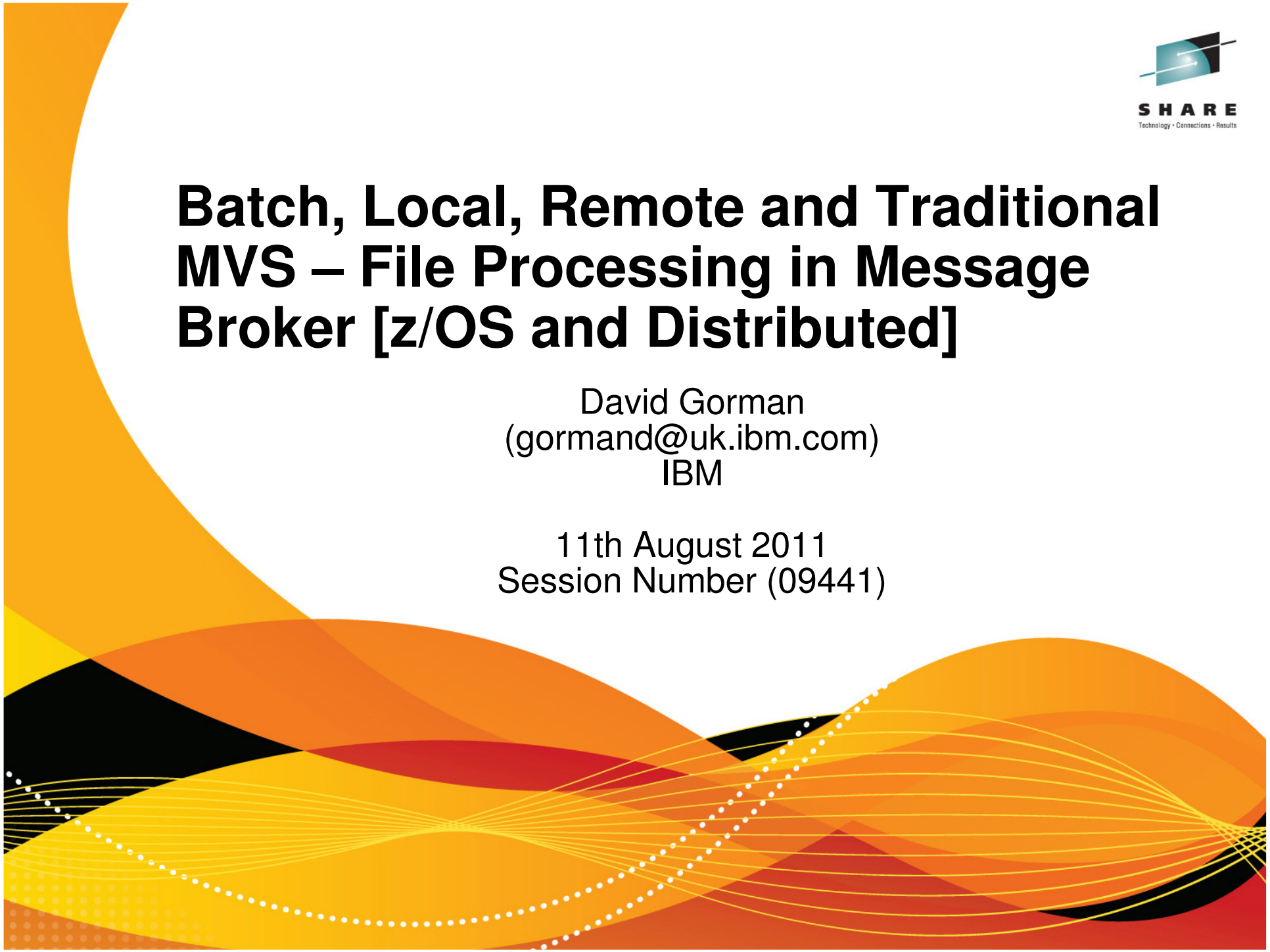


Batch, Local, Remote and Traditional MVS – File Processing in Message Broker [z/OS and Distributed]

David Gorman
(gormand@uk.ibm.com)
IBM

11th August 2011
Session Number (09441)

Agenda

- Integration using files
- File nodes
- FTE nodes
- CD nodes
- Additional SupportPac nodes
  - VSAM nodes
  - QSAM nodes

Intro

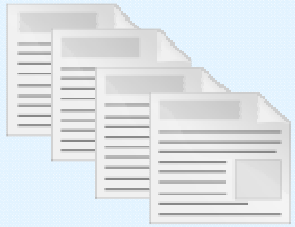
Organizations are frequently integrating file-based systems into an ESB as critical integration technologies, and WebSphere Message Broker (WMB) continues to extend its built-in support for files. WMB V7.0.0.2 adds a number of new features, including a new FileRead node that allows file processing to occur mid-flow.

WMB contains support for an embedded WebSphere MQ File Transfer Edition (FTE) agent that allows you to process files after their transfer, or trigger transfers on the back of processed messages. These facilities complement the existing file processing nodes which appear in the product.

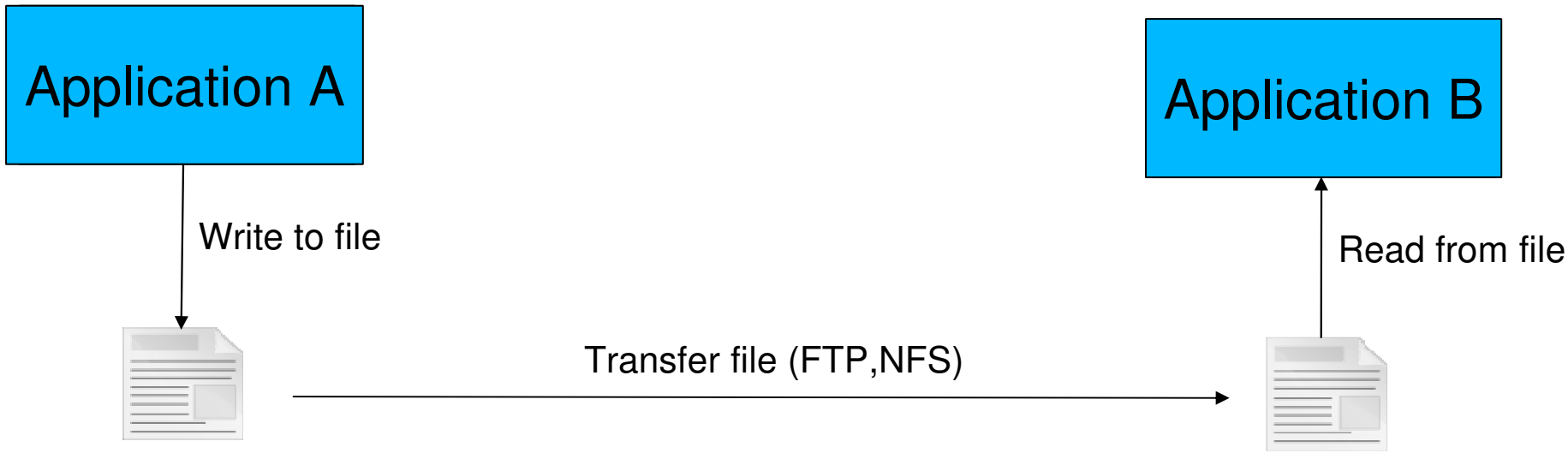
WMB also contains support for IBM Sterling Direct:Connect through new nodes introduced in APAR IC75621.

Support for VSAM and QSAM is also available via Cat3 support pacs IA11 and IA13.

Using files for integrating different systems



= information =$$



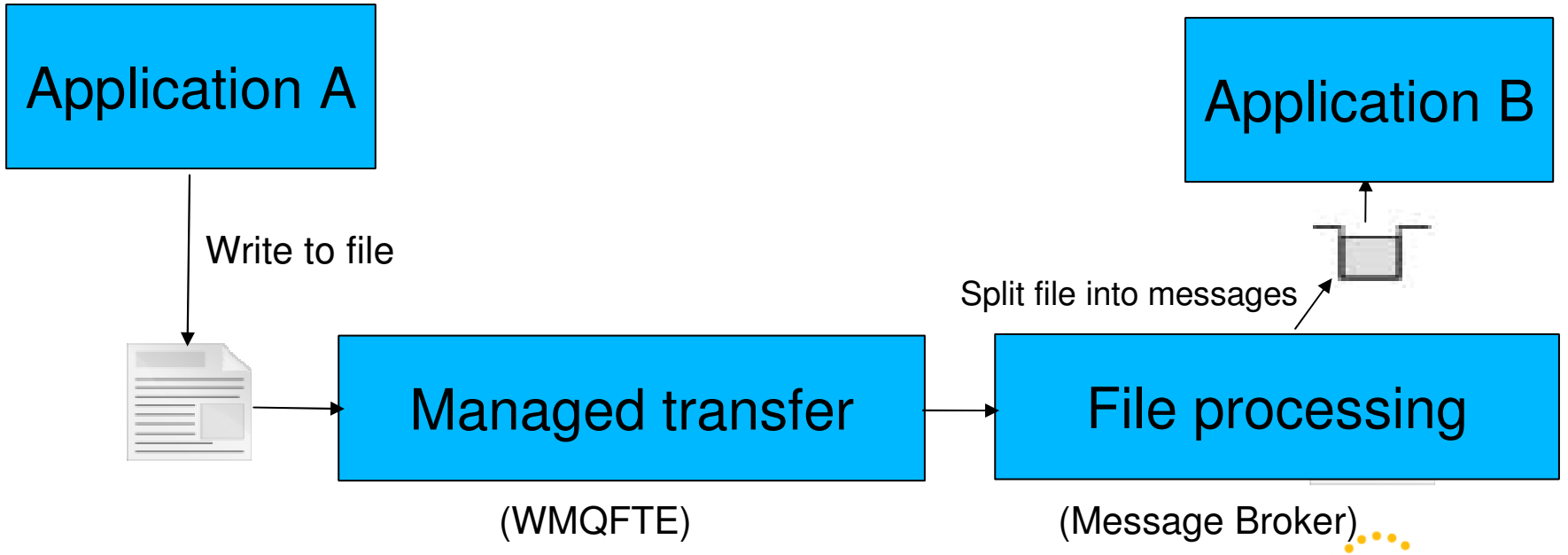
Notes : Using files for integrating different systems

- File based processing is still very common today even though we have had messaging software such as WebSphere MQ around for more than 10 years.
- We find that new applications often utilise the new technologies whilst existing applications tend to stay unchanged and utilise the original technology that they were based on. In recent year we have seen more new applications using files for sending and receiving data. Particularly in industries like retail.
- Files whilst very simple, mundane and lacking in glamour hold a lot of valuable information and so represent a lot of value and thereby money.
- Data in files is normally structured and will have meaning once we can understand the content and structure.
- General flow of a file integration is:
  - Application A at a given time of day exports a set of data to a file
  - FTP is used to transfer the file across either as a manual task or using a script that triggers at a given time. NFS file systems are also some times used so two apps can share the same file
  - Application B at a given time of day imprts the file transferred from application A.
- The files tend to be batches of records rather than single messages like when using WebSphere MQ.

The world moves ...

New protocols:
- WebSphere MQ
- HTTP
- JMS

New processing models:
- Real time
- Event processing
- Request/reply

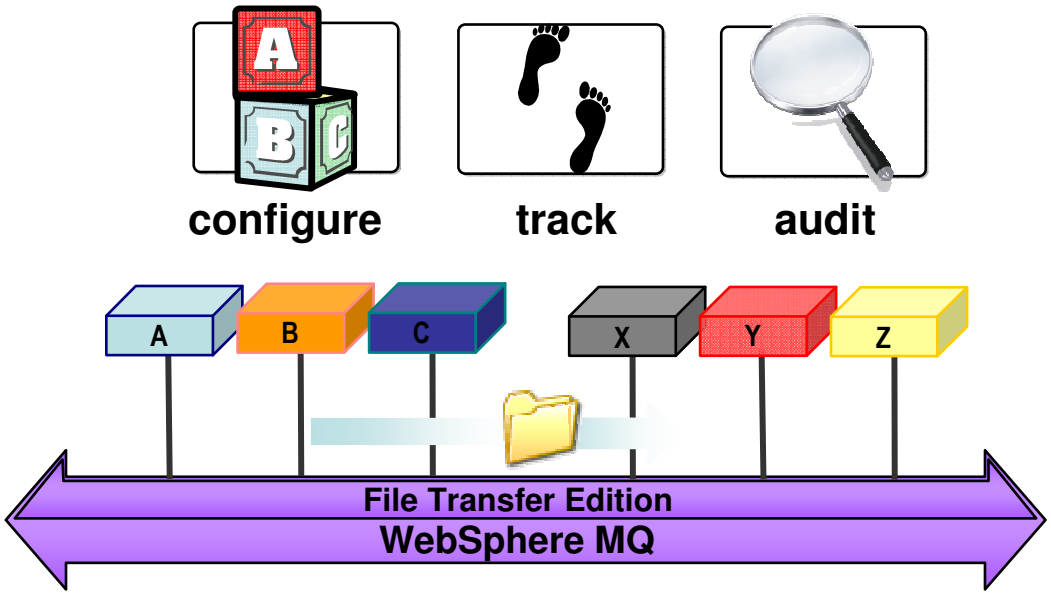


Notes : The world moves ...

- New protocols and processing models are introduced to businesses.
- Tend to be message based and real time and event driven. Often request reply models are used where a piece of data sent requires a response at a given time.
- File based integration is an alternative and has strengths as well as weaknesses.
- Some times batch processing fits the processing model within a company and is easy to understand by administrators.
- To have the best of both both worlds then use products to managing the delivery and the integration with new protocols.
- Two main parts or phases for file integration:
  - File delivery
  - File processing

What is Managed file transfer

- There is no standard that dictates what Managed File Transfer is and isn't
- Software that overcomes the problems inherent with traditional methods of transferring files
- WebSphere MQ File Transfer Edition is a Managed File Transfer Solution



- WMQFTE is a standalone product but can also be run from within Message Broker

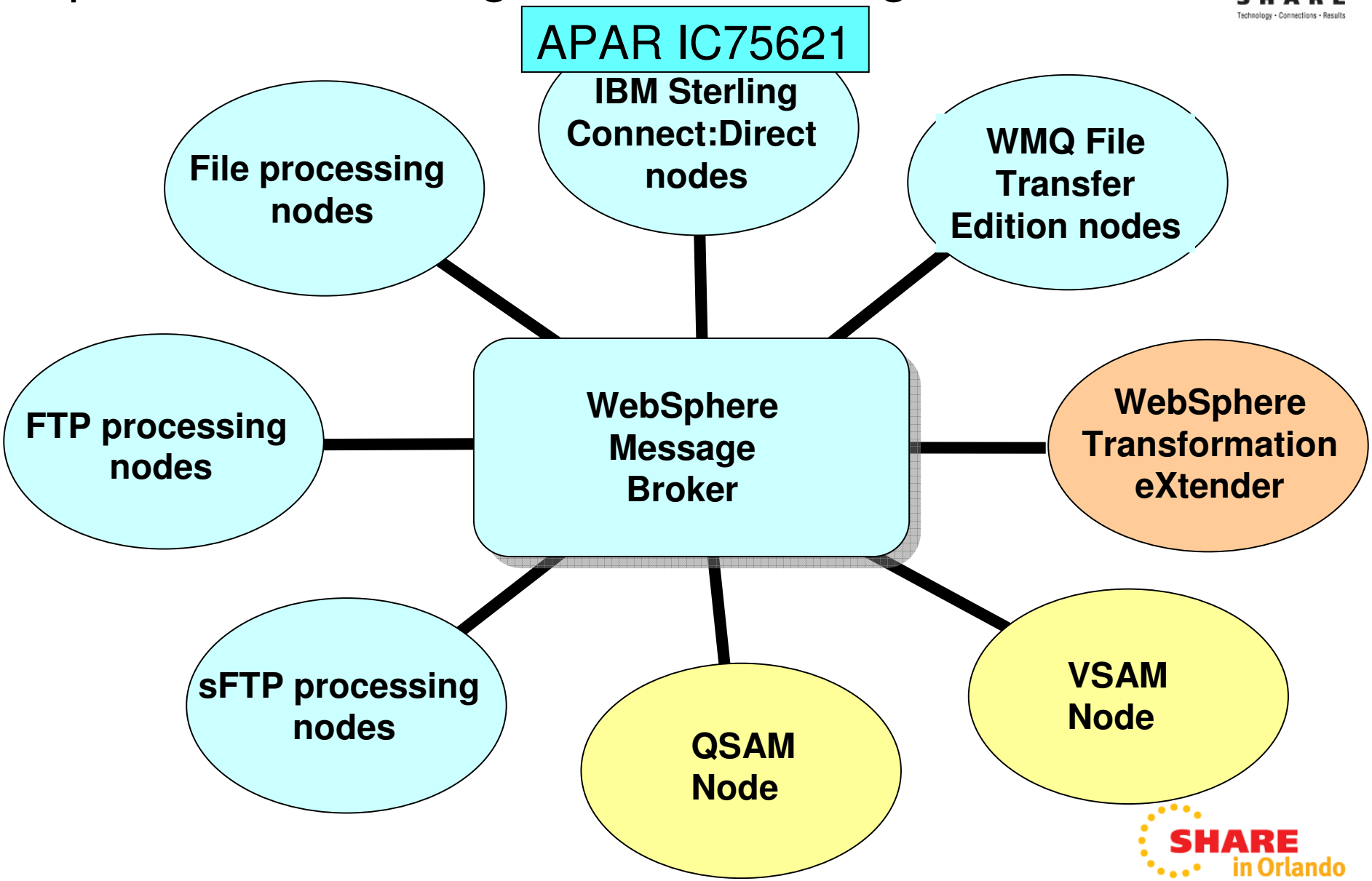
Notes : What is Managed file transfer

- Three main areas that are critical for file delivery:
  - Configure – Make it is easy to set up and automate the transferring of files based on triggers like time of day or a file arriving in a given directory,
  - Track – track the file being transferred from on system to another and being able to identify where a problem has occurred.
  - Audit – have a historical record of what file were transferred, when it happen and what the status of the transfer was.

What is File Processing

- File processing is:
  - **Splitting** a batch of records within a file to a set of messages to process
  - **Combining** messages to produce a batch of records in a file
  - **Transforming** data into different formats
  - **Routing** based on file content
  - **Connecting** and sending to different protocols:
    - MQ, JMS, HTTP, Raw TCPIP, Databases …
- WebSphere Message Broker is the perfect fit with the following core values:
  - Universal Connectivity
  - Routes and Transforms data FROM anywhere TO anywhere
  - Simple programming
  - Many environments
  - High performance
- WebSphere Message Broker is tightly integrated with file delivery solutions
  - WMQ FTE embedded agent
  - FTP and SFTP
  - IBM Sterling Connect:Direct
  - VSAM & QSAM

Options for file integration in Message Broker



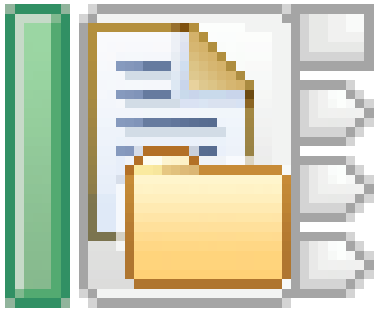
Notes : Options for file integration in Message Broker

- Message Broker has comprehensive file integration options:
  - The blue circles show built in nodes which come as apart of the standard message broker install and require no additional products or install steps.
  - The yellow circles are cat 3 support pacs which provide QSAM and VSAM function and can be added to the base Message Broker install
  - The orange circle is the the **WebSphere Transformation eXtender**

Agenda

- Integration using files
- File nodes
- FTE nodes
- CD nodes
- Additional SupportPac nodes
  - VSAM nodes
  - QSAM nodes

File input node



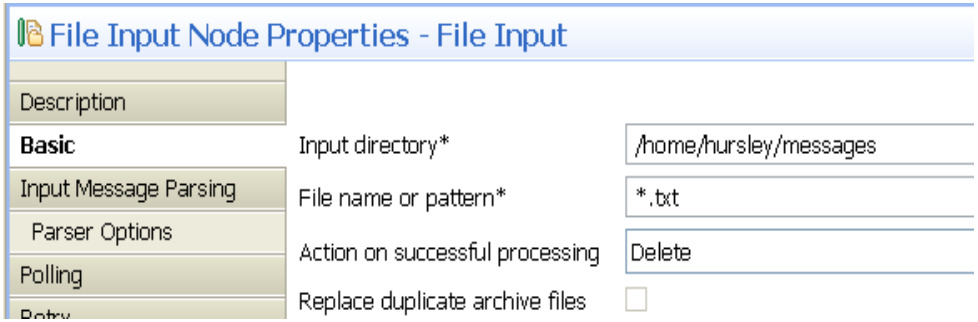
File Input

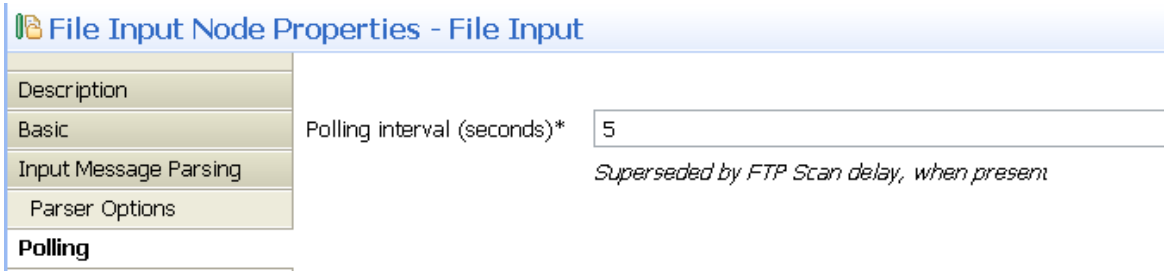
Notes : File input node

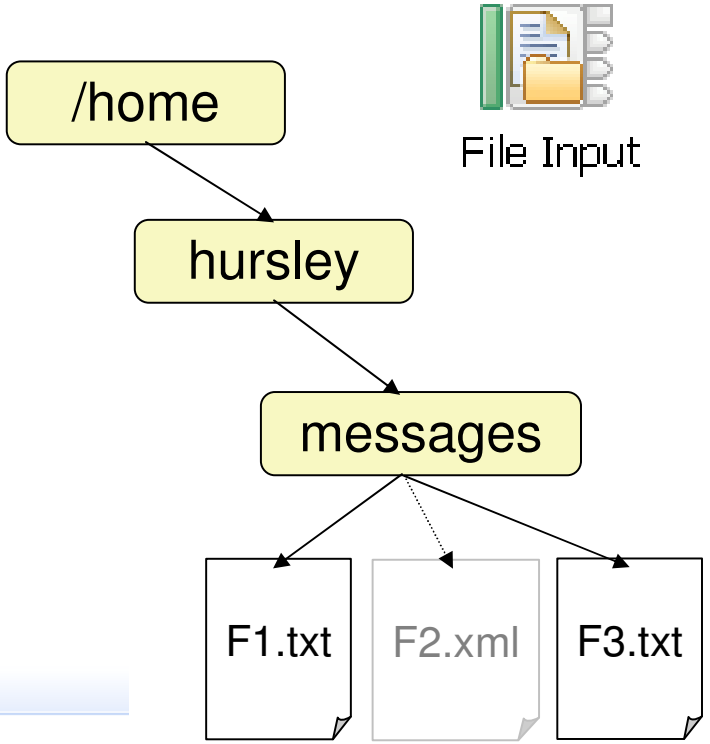
- The input nodes reads data from a file and triggers the start of processing in the flow. Three main areas will be covered in detail:
  - The mechanism used to detect a file is ready to be processed
  - The splitting of the file up into records
  - The archiving or deleting of the file once processing has been finished

FileInput node – Operation

- Scans a pre-configured directory (relative or absolute) for files that match a given specification
- Locked files are ignored until they become unlocked







FileInput node – Record Detection

- Handling options (on the Records and Elements tab):
  - Whole file
  - Fixed Length *
  - Delimited *
  - Parsed Record Sequence *

Note * - results in separate records - message flow is invoked multiple times

- Only requires one record to be in memory at any one time
  - Allows very large files (Gigabyte) to be streamed efficiently
  - Streaming possible with MRM (CWF and TDS) and XMLNSC parsers only

- If connected, ‘End Of Data’ terminal is triggered at end of file
  - Empty BLOB message and a LocalEnvironment.File structure

Record Detection Examples

- With input file `Belgian Bun|10|Jam tart|9|Gingerbread man|8|`

File Input

- Whole file
  - Propagates one message:
    - `Belgian Bun|10|Jam tart|9|Gingerbread man|8|`
- Fixed length (size = 10 bytes)
  - Propagates five messages:
    - `Belgian Bu` `n|10|Jam t` `art|9|Ging` `erbread ma` `n|8|`
- Delimited (character = '|')
  - Infix: propagates seven messages
    - `Belgian Bun` `10` `Jam tart` `9` `Gingerbread man` `8` `""`
  - Postfix: propagates six messages
    - `Belgian Bun` `10` `Jam tart` `9` `Gingerbread man` `8`

Record Detection Examples 2

- With input file:

```
<cakes><cake name="belgian bun" rating="10"></cakes>
<cakes><cake name="jam tart" rating="9"></cakes>
<cakes><cake name="gingerbread man" rating="8"></cakes>
```

- Parsed record sequence
  - When the parser is set to XMLNSC, this propagates three XML messages

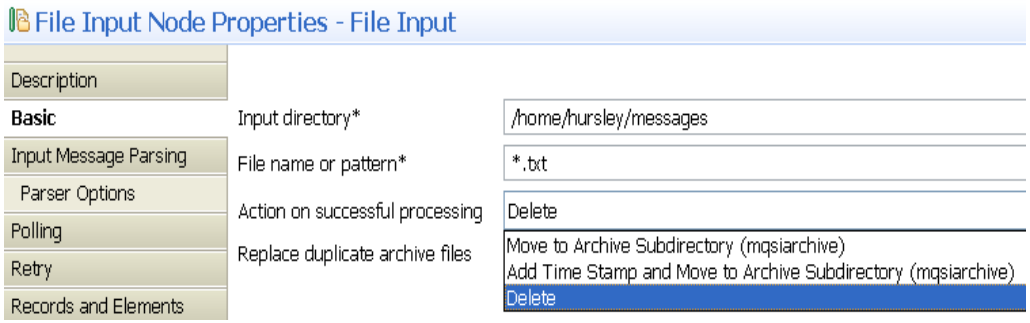
```
<cakes><cake name="belgian bun" rating="10"></cakes>
```

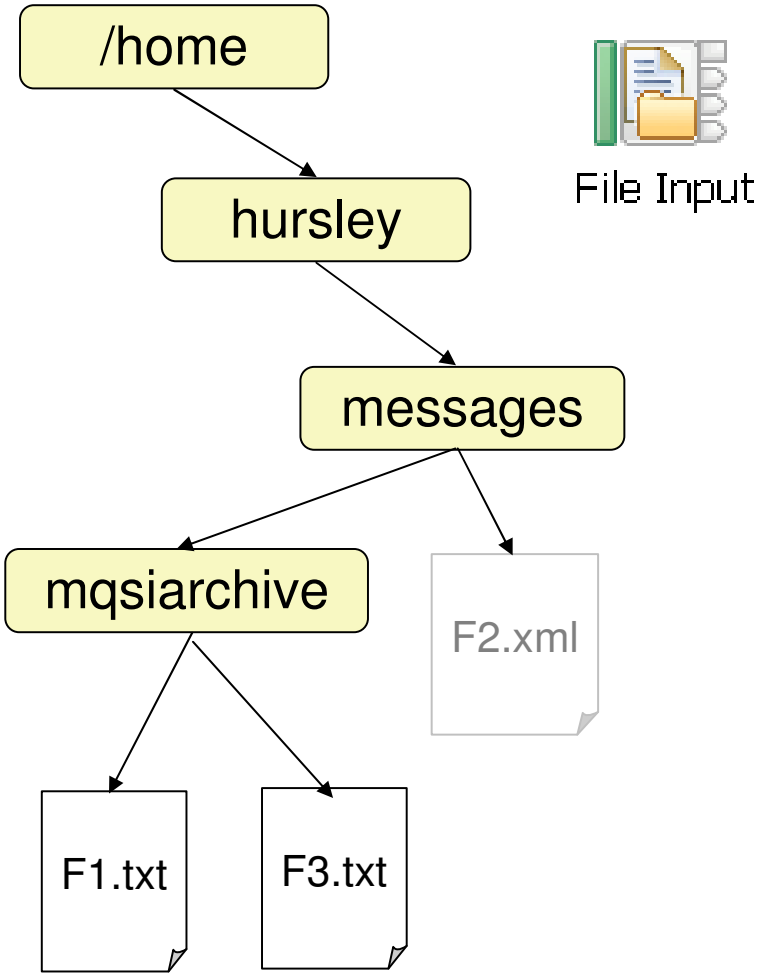
```
<cakes><cake name="jam tart" rating="9"></cakes>
```

```
<cakes><cake name="gingerbread man" rating="8"></cakes>
```

FileInput node – Archiving

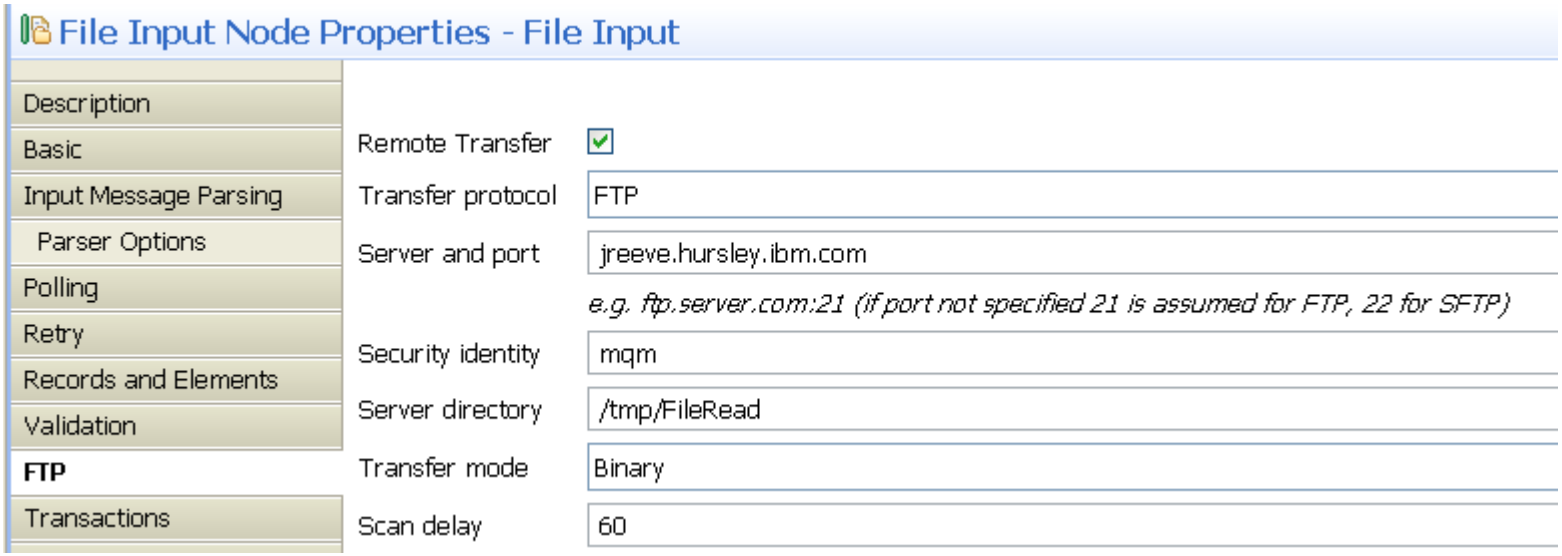
- Upon successful processing, file is either deleted or moved to an *mqsiarchive* subdirectory
- Dealing with files with duplicate names:
  - Option to include timestamp in archived filename
  - Option to replace any existing file





FileInput node – using FTP and SFTP

- When active, FTP settings cause the node to periodically transfer files on a remote server to the local directory for input.
- Security Identity ‘UserMapping’ set using runtime command:
  - mqsisetdbparms BROKER –n ftp::UserMapping –u USER –p PASS



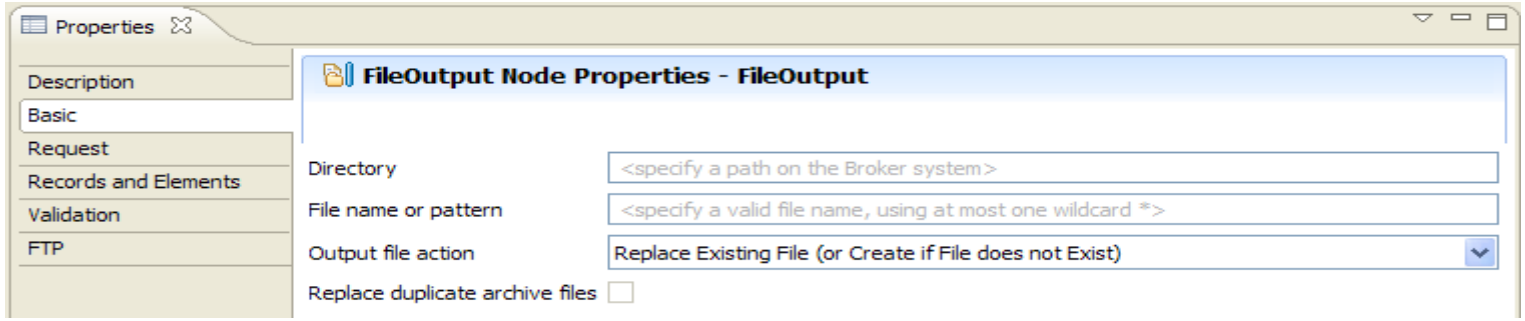
File output node

Notes : File output node

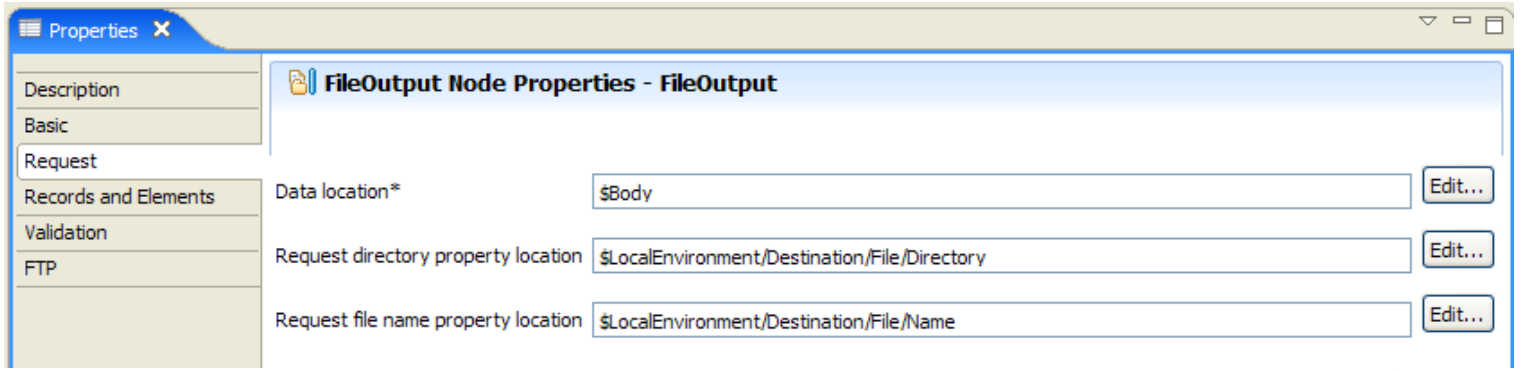
- The file output node is used to create and write to a file anywhere in the middle of a flow:
  - How it decides which file to create and where
  - How a file is created using a series of records appended
  - What happens if the file it attempts to create already exists

FileOutput – Writing files

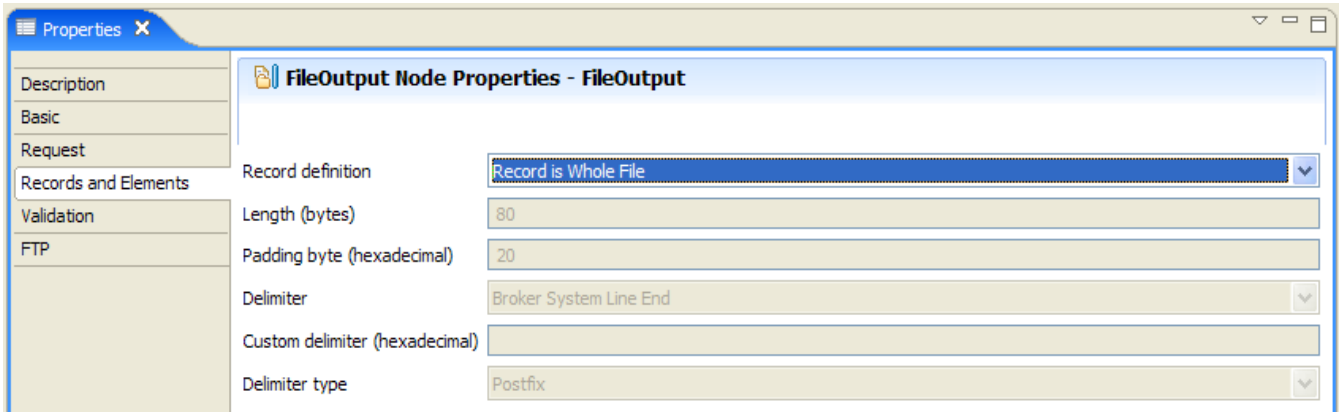
- In the simplest scenario, the received message body is written to the pre-configured file:



- When writing to the output file, the wildcard (if present) is replaced with the value of *LocalEnvironment.Wildcard.WildcardMatch*
  - Allows you to preserve elements of a filename during processing



Appending Records



- "Records and Elements" tab defines how multiple writes to the same file are handled
- Record definition options:
  - Record is Whole File – close file automatically after first write
  - Record is Unmodified Data – the message bit-stream appended to file
  - Record is Fixed Length Data – specify length in bytes and padding character
  - Record is Delimited Data – specify delimiter and infix/postfix option
- Unless "Record is Whole File" is selected, the file will be closed when the "Finish File" terminal is triggered
  - The Finish File message is propagated on to the FileOutput's "End Of Data" terminal
- The *mqsitransit* subdirectory holds all files that have not yet been closed

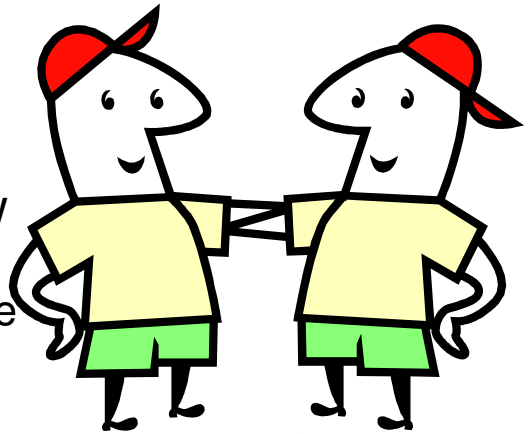
Options If The File Already Exists

- Output file action (basic tab)

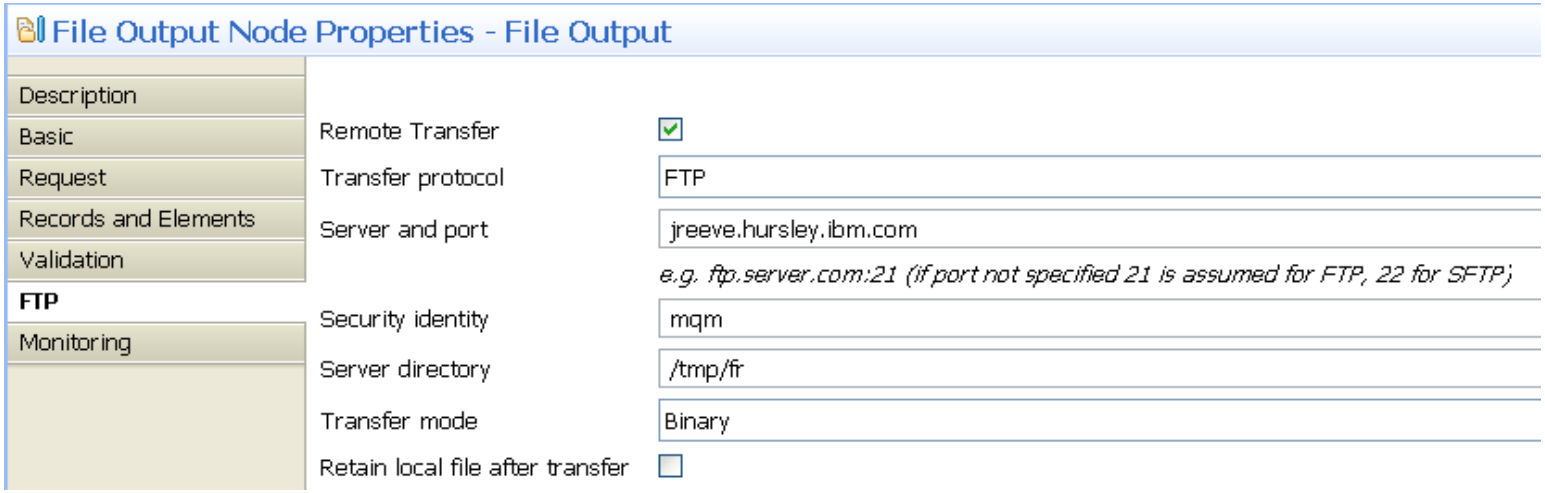
Replace Existing File (or Create if File does not Exist)
Create File (Fail if File Exists)
Archive and Replace Existing File (or Create if File does not Exist)
Time Stamp, Archive and Replace Existing File (or Create if File does not Exist)

- If the file already exists
  - Replace it
  - Go down failure terminal
  - Move to mqsiarchive subdirectory
  - Add timestamp and move to mqsiarchive subdirectory

- Each file can be built up over multiple message flow invocations (i.e. have many records)
  - However, once a file has been closed, it cannot be appended

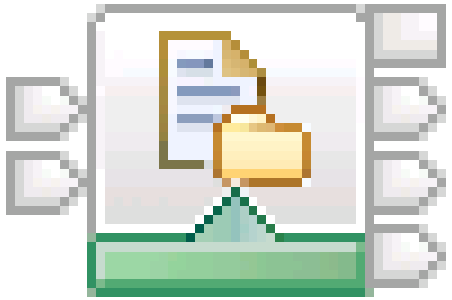


FileOutput - FTP Support



- If enabled, whenever a complete file is closed an FTP transfer of the file is attempted to the supplied FTP server
- File is optionally deleted from the local file system when the transfer completes
- Transfer is synchronous
  - Use additional instances if throughput rate is an issue

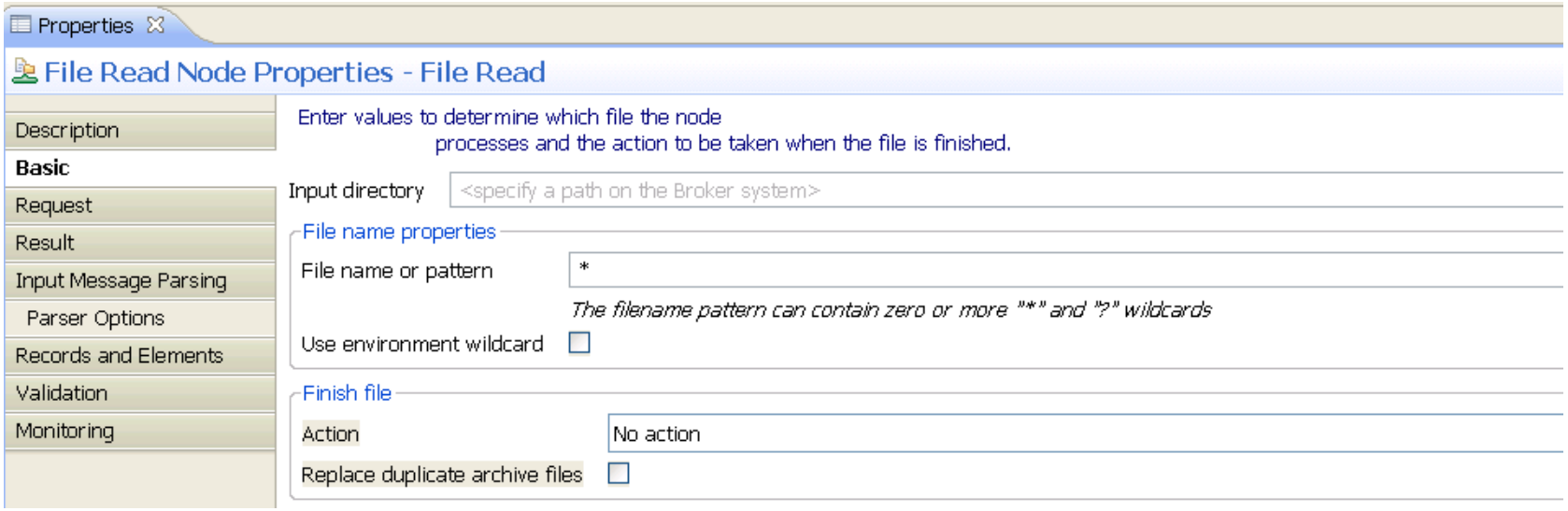
File read node



File Read

New FileRead node

- Introduce a new File Read Node
  - Reads data in middle of flow like an MQGet or a TCPIPReceive node
  - Reads either whole file contents or one record from the file
  - Allows user to override the file to be read and the offset within the file to start reading from
- Has a *No match* terminal which the message is sent to if it can not find a file or a record in the file

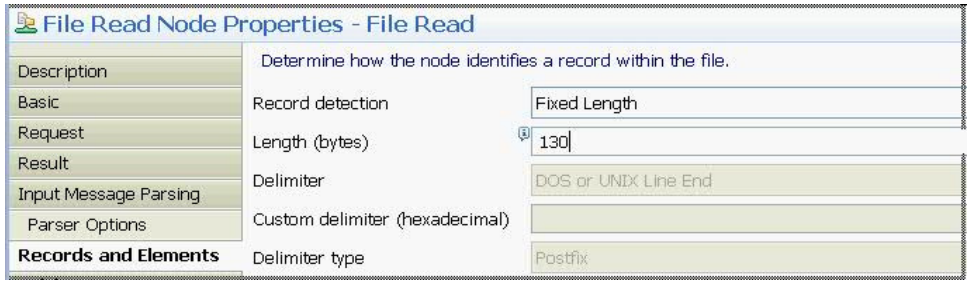


Notes : New FileRead node

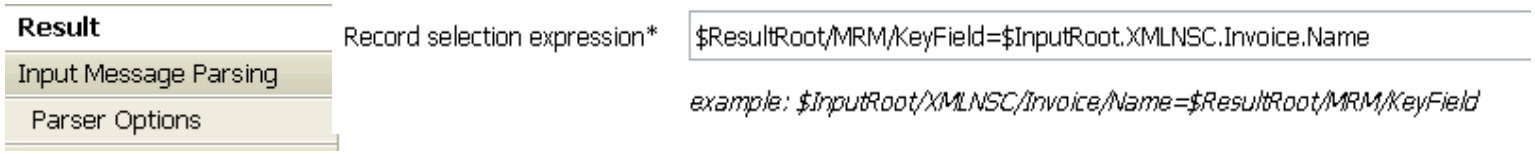
- A new file node which behaves like a MQGet or TCPIPReceive node in the sense that it reads in data within a flow without first sending data out. For example: MQGet reads a message from a queue, TCPIPReceive reads data from a TCPIP input stream and the Fileread node reads data from a file.
- Can either read the whole contents of a file or a single record from the file and then parses and constructs a message to propagate down the flow.
- The node is very configurable both at design time and during runtime where most properties can be overridden based on the message content.
- The basic properties are similar to a File input node where the details of the file to process are given.

Defining which record to read and propagate

- **Where the record starts**
  - Defaults to the beginning of the file
  - Give the offset into the file to start record from based on contents of the message
- **Where the record ends**
  - Define the record detection mechanism:
    - Fixed size
    - Delimited
    - Parser



  - Give the length of the record based on the contents of the message
- **Which record to propagate**
  - Define an expression to specify which record to propagate
  - Node iterates through all records from the start offset until one matches
  - Only propagates the first match

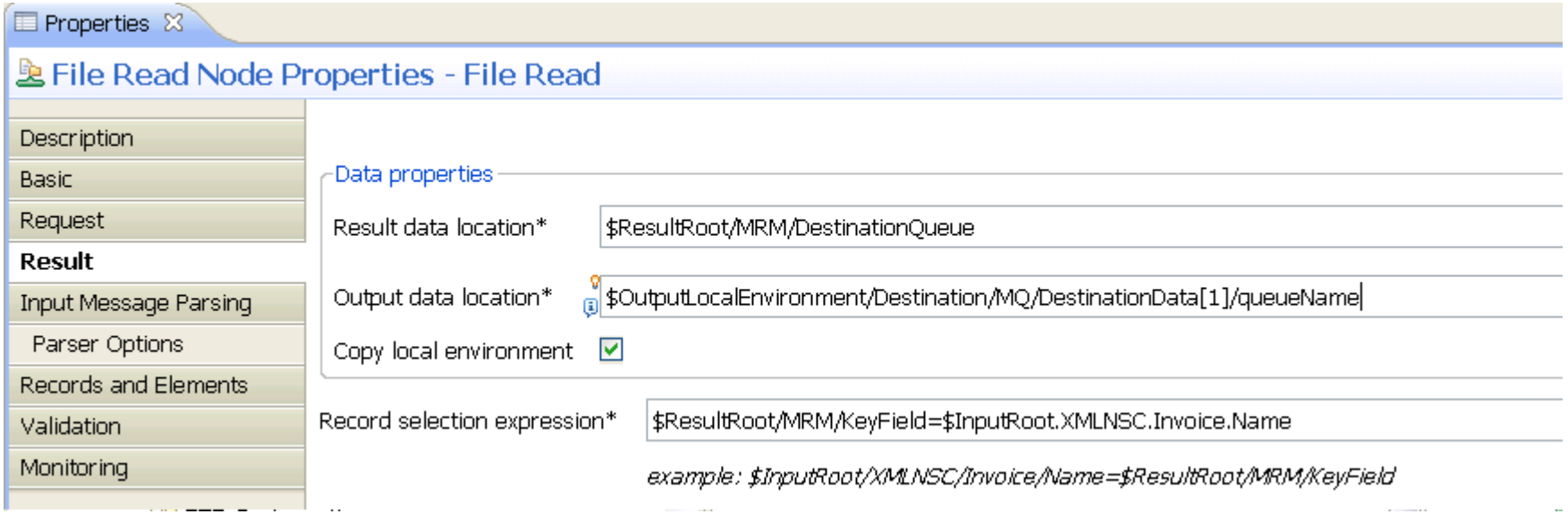


Notes : Defining which record to read and propagate

- Three key pieces of information need to be defined to create and propagate a record from the file.
- Where the record starts. By default the file read node always starts the reading of a record from the beginning of the file. It does not remember or store the result of the last record read so the next time through the node will start from the same place. Unless the user specifies in the local environment where to start from. It is possible to configure the node to use the end record offset from a previous fileread node as the start of the current record.
- Where the record ends. The normal record detection mechanisms are used to find the end of the record. Fixed size just reads that many bytes, delimited scans for a delimiter and parser uses a message broker parser to determine the end (like XMLNSC if the record is an XML document). When using fixed size it is possible to override the length being used using the message content or local environment.
- Which record to propagate. After finding the record the *Record selection expression* is evaluated. If it is true then the record is propagated otherwise the next record is found using the end of the last record as the start of the new one. The process is repeated until either the expression is true or the end of file is found. Only the first matching record is propagated. If no record matches then the file gets sent to the no match terminal.

Constructing the message using the record read

- **Which part of the record read to propagate**
  - Result data location
- **Where to put the record in the outgoing message**
  - Output data location

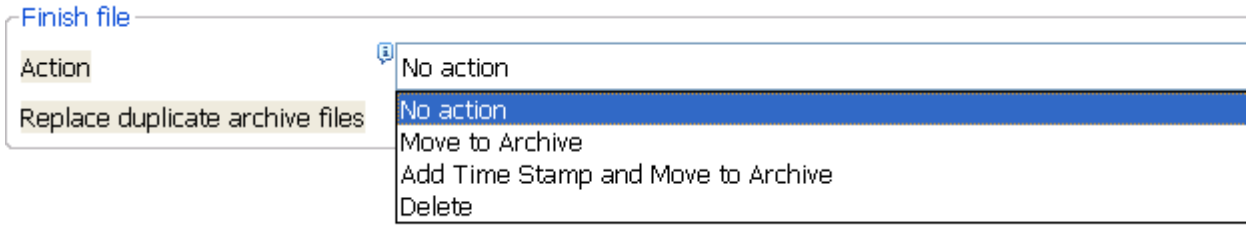


Notes : Defining which record to read and propagate

- The *Result* panel has properties which specify how the outgoing message is constructed based on the contents of the file and the incoming message.
- By default, the whole incoming message is replaced with the contents of the record retrieve from the file.
- The Result data location is used to extract a piece of information from the file to insert into the outgoing message.
- The Output data location is used to find the location to write the extracted information to.
- For example:$ResultRoot/MRM/DestinationQueue extracts the value of the DestinationQueue field and then writes it to a copy of the incoming message assembly at the location $OutputLocalEnvironment/Destination/MQ/DestinationData[1]/queueName

Changing the file disposition after the read

- By default file is left unchanged after read
- Disposition change is always done when the FileRead node executes *Finish File*:
  - The end of the file is reached
  - A message is sent to the finish file terminal
- Following actions are available:



- Archived files are moved to the mqsiarchive directory
- Can override the archive directory and archive name using local environment or data in the message

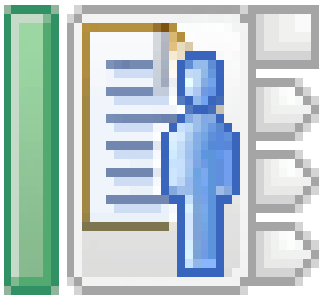
Notes : Changing the file disposition after the read

- By default, after any read, the file read node will leave the file unchanged and will close any connections to it.
- It is possible to configure the node to modify the file disposition when the nodes *finish file* action is triggered. Finish file is defined as either when the file read node reads to the end of a file (whole file mode or when the last record is read) or when a message arrives at the *finish file* terminal.
- It is possible to delete or archive the file. Archiving moves the file to the mqsiarchive directory but it is possible to override this based on the contents of the message to move the file to any new directory and any new file name.

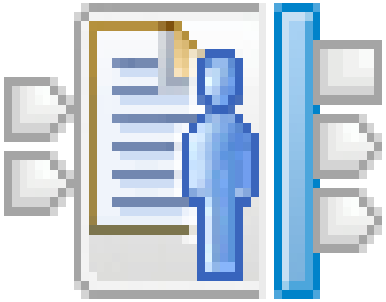
Agenda

- Integration using files
- File nodes
- FTE nodes
- CD nodes
- Additional SupportPac nodes
  - VSAM nodes
  - QSAM nodes

FTE Support in Message Broker 7.0.0.1

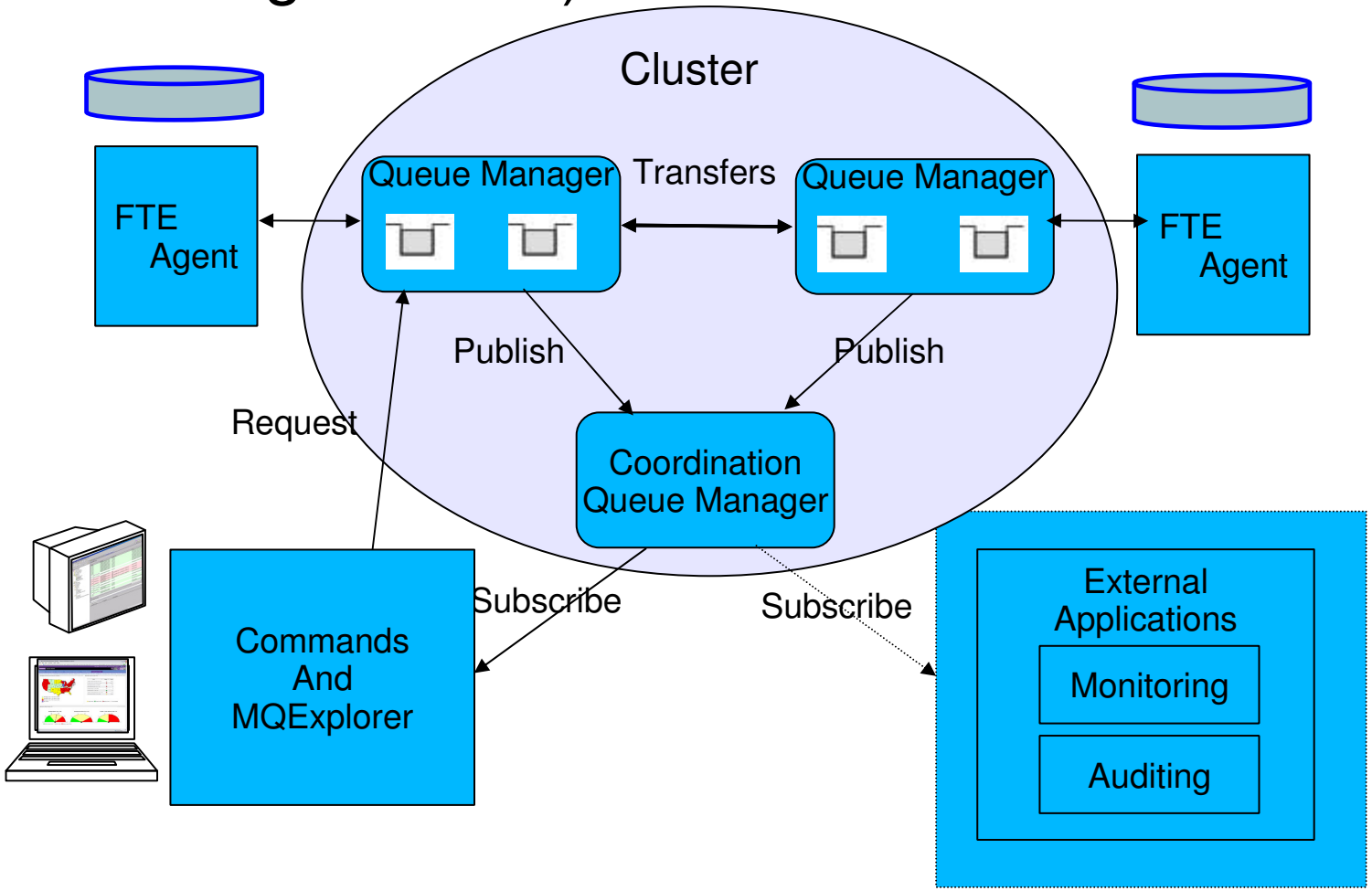


FTE Input



FTE Output

WebSphere MQ File Transfer Edition Overview (no Message Broker)



Notes

This is a view of how a standard FTE network looks like without message broker being involved.

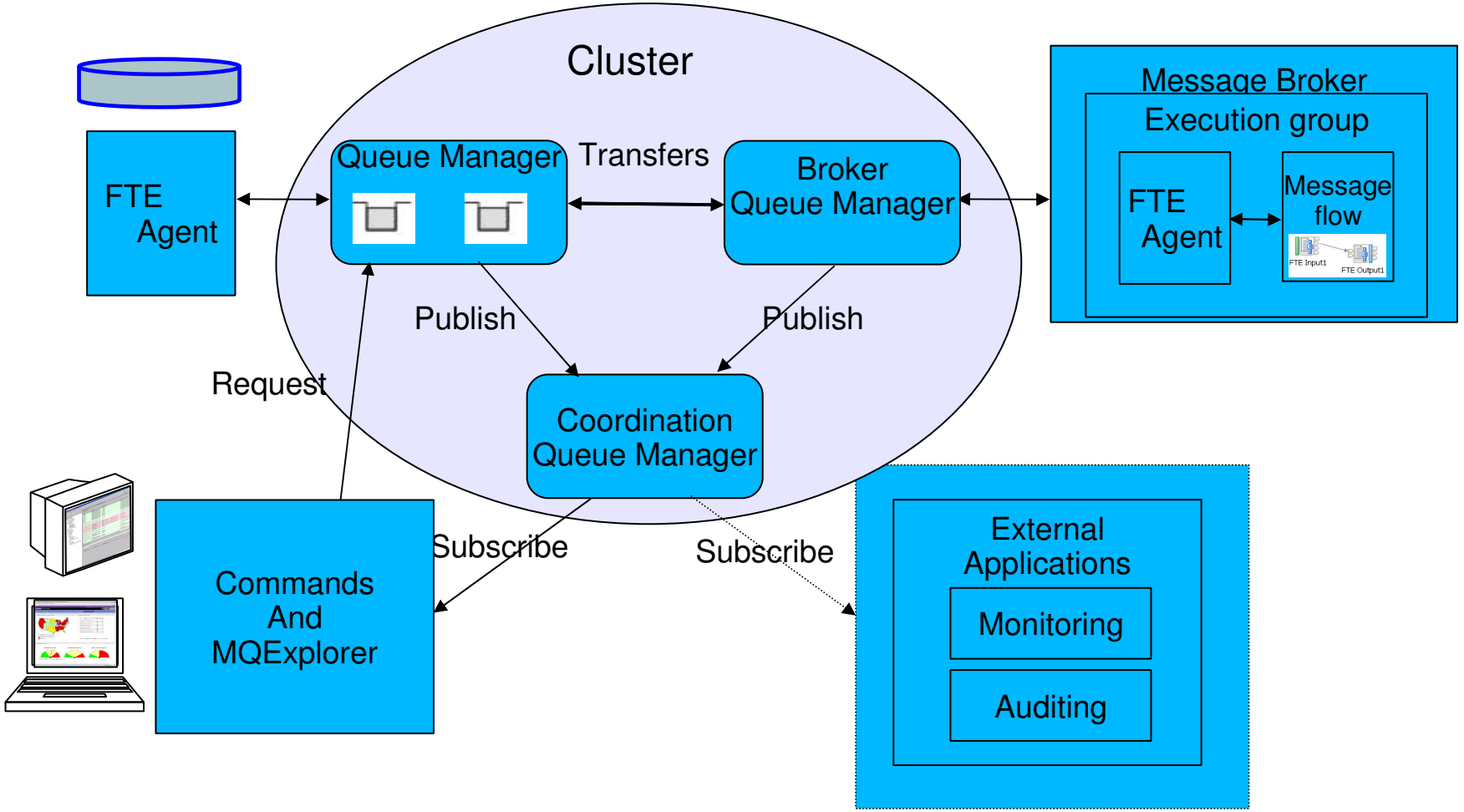
FTE agents transfer files between themselves using WebSphere MQ.

The agents directly transfer files by sending a series MQ message from a local agent on one system to a remote agent on another system. The messages are routed using standard MQ queue manager mechanisms. This requires all the agent queue managers to have channels set up correctly so that each agent can access a remote agent via their local queue manager. The simplest way to achieve this is by adding all the queue managers involved into an MQ Cluster.

As well as directly sending the file data to the remote agent each agent also publishes audit messages to a coordination queue manager. Other applications can then subscribe to these published messages to monitor how the transfer has progressed. The coordination queue manager being down does not cause the direct transfer to fail (unless it is also a queue manager required to access the remote agent).

Tools like MQExplorer can be used to monitor the progress of any transfer

Adding Message Broker to an existing FTE network



Notes : Adding Message broker to an existing FTE network

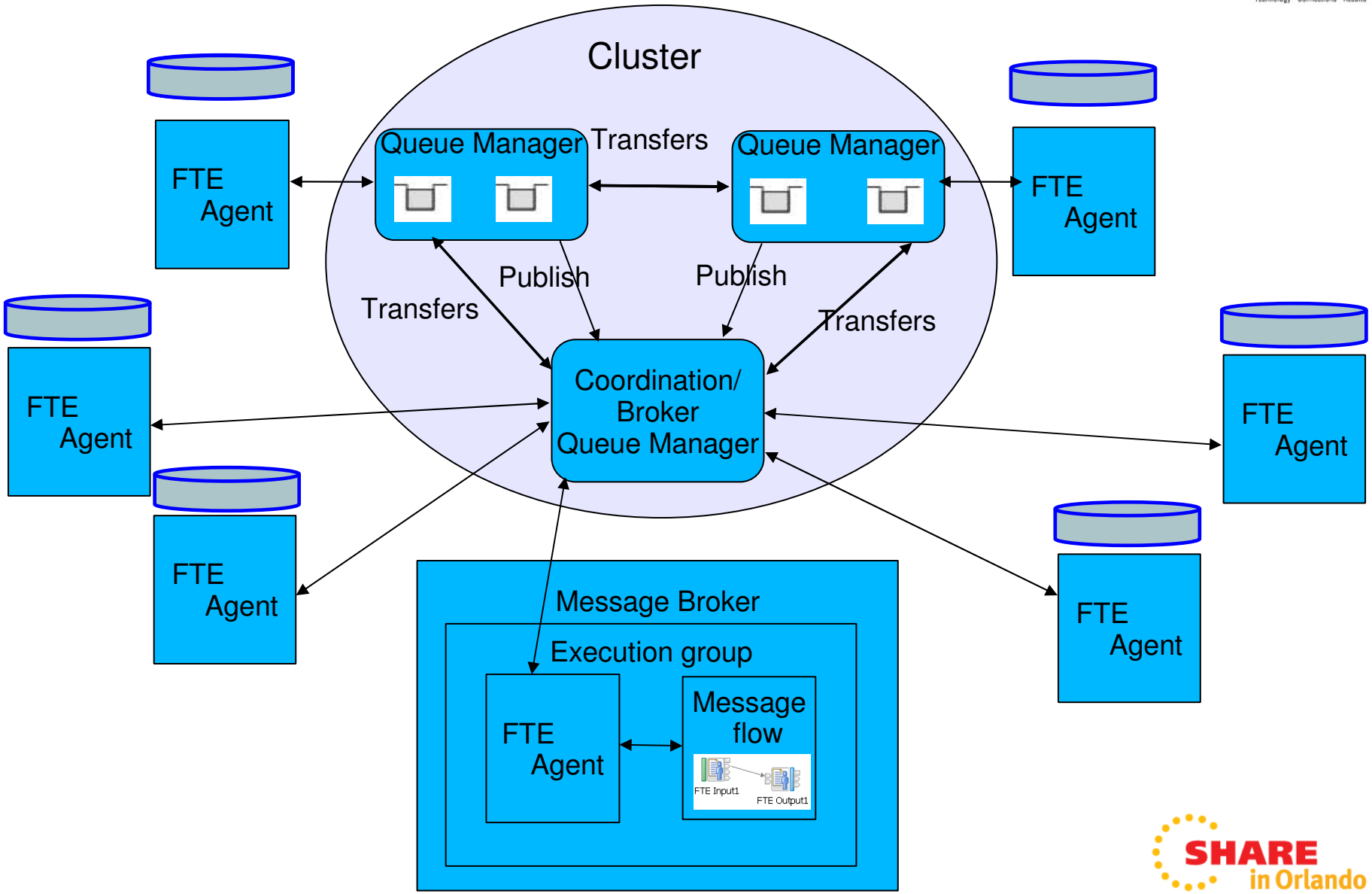
In version 7.0.0.1 it is now possible to replace any of the agents in an existing network with an instance of message broker.

The message broker runs an agent embedded within a message broker execution group which behaves just like a standalone FTE agent.

It is possible to add message broker to existing agent networks or to construct new networks which have a combination of broker agents and standalone agents.

In fact it is possible to construct an entire FTE network which only has broker embedded agents.

Creating a FTE network around Message Broker

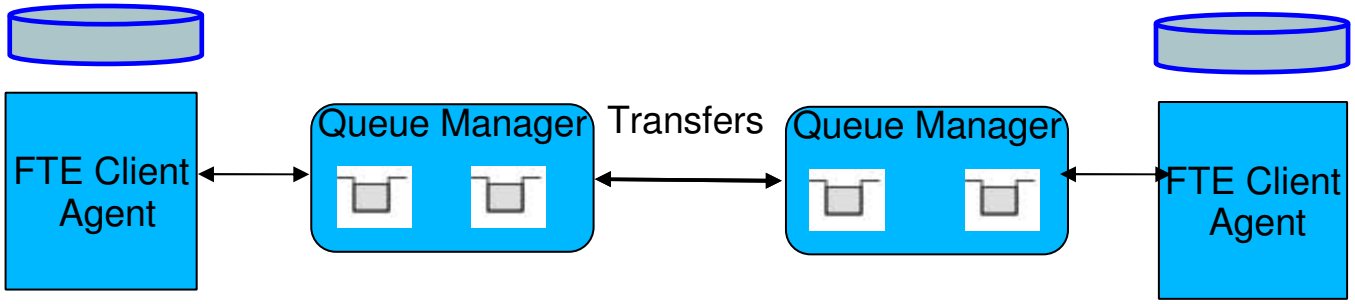


Notes : WebSphere MQ File Transfer Edition Overview (no Message broker)

Another approach when constructing a FTE network when using message broker is to have message broker as a central agent on the coordination queue manager and have many standalone FTE agents on other machines. With this design the standalone agents can either directly transfer their files to other agents or transfer them to the central broker agent which could do any required processing before transferring it on to another agent or, if required, sending it to another system using a different transport protocol like HTTP or FTP. It would also allow for transformation of data before being sent to an end destination.

WebSphere MQ File Transfer Edition Runtime Components

- WebSphere MQ File Transfer Edition uses a peer to peer architecture to transfer files
- There is **No** central server agent with client agents connecting in to it
- Two versions of WebSphere MQ File Transfer Edition product sold:
  - WebSphere MQ File Transfer Edition Client – Client MQ connections
  - WebSphere MQ File Transfer Edition Server – Client or server MQ Connections
- It is possible to transfer a file between two Client agents without going through a Server agent.



- WebSphere Message Broker embeds the Server version of the agent.

Notes : WebSphere MQ File Transfer Edition Runtime Components

The FTE product is sold in two versions called Server and client which have different license requirements. From a technical point of view the main difference is that the server version can connect to MQ either using client or server bindings but the client can only use client connections.

Despite the name client and server this is no client agent connecting to server agent. The only real server that is involved is the WebSphere MQ queue manager and it is completely possible for one client agent to transfer to another client agent without any server agent being involved.

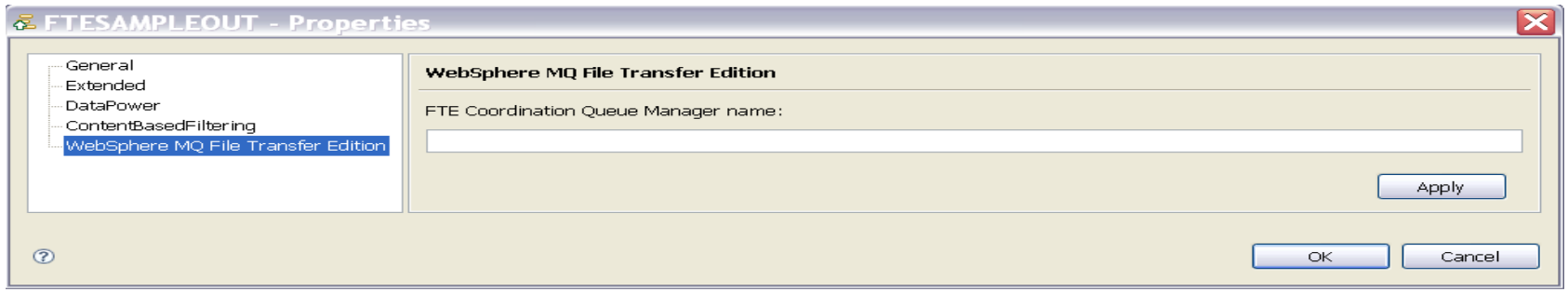
The message broker embedded agent runs a Server copy of the agent and always uses server bindings to the local broker queue manager. It is not possible for it to run in a client connection mode.

The broker agent can not run any user exists or run any commands/ant scripts. It also does not include the FTE protocol bridge (but it can do all the normal protocol bridging that it using the various built in message broker nodes).

The agent that is included in the message broker product is included under the standard message broker licence and does not need any additional licenses. Any other FTE agent running outside message broker needs it own license.

Creating and running an agent in Message Broker

- Install
  - FTE code is installed as part of broker install. No need for FTE to be installed separately
- Deployment
  - Agents run in the execution groups' JVMs. One agent per execution group
  - Coordinating queue manager defined as an execution group property



  - Agent name is derived from broker and execution group name:
    - <broker name>.<execution group name>
  - FTE Agents are created automatically by broker including required queues
- Starting/stopping
  - Agent started when first node using it is deployed
  - Agent stopped when last node using it is un-deployed or stopped

Notes : Creating and running an agent in Message Broker

The core FTE product is installed as part of the normal message broker install. No additional components need to be installed apart from the MQExplorer add on which can be used to monitor transfers. It is not required for the transfers to work but is useful for monitoring what is happening with transfers.

Each execution group with in broker can be configured to run an FTE agent. This is done by setting the coordination queue manager property on the execution group either using MBExplorer or the mqsichangeproperties command.

The agent name is derived by concatenating the broker and execution group name together. If the name is too long to be a valid agent name then it is truncated. If it contains non-valid characters (any characters not supported in MQ queue names) an error is written to the local system log.

All the configuration for the agent is created when the coordination queue manager is set and also deleted when it is unset.

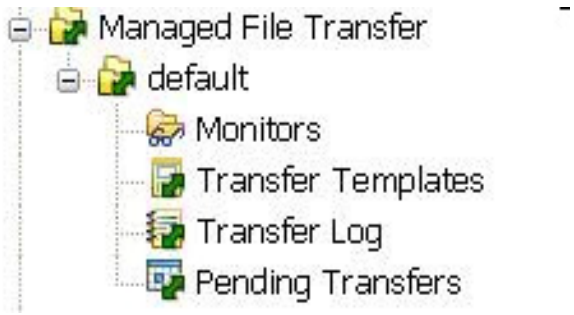
As well as creating the required config files, all the required queues are also created.

The actual agent is started when the first node using it is started and stopped when the last node using it is stopped.

If a node is deployed with out setting the coordination queue manager the an agent will be created at deployment time using the brokers queue manager as the coordination queue manager. This is a temporary agent which is deleted when the last node using it has been stopped. It is recommended that the execution group property is set even when the broker queue manager is the coordination queue manager.

Administering an agent in Message Broker

- Configuration
  - Agent configuration is created under the broker WORKPATH
  - <WORKPATH>/components/<broker>/<execution group>/config/WMQFTE
  - Identical files and structure to standalone Agent
- Logging and tracing agent
  - Log still written to config directory
  - Log also written to user trace and trace written to service trace
- FTE transfer directories
  - Default directory and staging directories created under broker WORKPATH
  - <WORKPATH>/common/FTE/<broker>/<execution group>
  - Default transfer directory under Inbox
  - Staging directory for Output nodes under Transfers/<flow name>/<node name>
- Monitoring transfers
  - Use MQExplorer plugin



Notes : Administering an agent in Message Broker

The administration of the agent is very simple with almost all tasks being done automatically by the execution group.

All the config files created are identical to those used in a standalone agent.

Logging by the agent is written to the standard broker user and service trace but the log file in the config directory is still written to as well.

A location under the workpath is used as the default transfer directory and also to stage files before they are transferred To separate files being transferred by different nodes a directory structure is created which includes the flow name and node name.

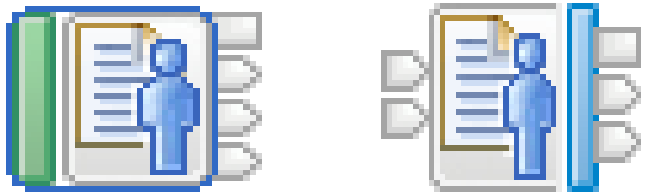
As with the file nodes a mqsitransmit directory is used to build files up before transferring them. The names of files in these directories are mangled to stop clashes of files with the same name which are going to be transferred to different agents or different directories. If is recommended not to delete files from the transmit directory unless the message flow using them has been stopped and all files are deleted.

The FTE MQExplorer plugin is very useful to monitor the progress of any transfers which have been done.

Accessing agent from a message flow

- Messages received and sent from agent using node:
  - FTEInput node: receives any file transferred to execution groups agent
  - FTEOutput node: constructs a file and sends a request to the execution groups agent to transfer a file
- New FTE nodes but based on the current file nodes allowing:
  - Record based processing
  - Stream parsing for large file support
  - File filtering
  - Can be used with all other Broker nodes

- Agent is started or stopped based on whether any nodes are running



Notes : Accessing agent from a message flow

Message broker nodes are used to interact with the embedded agent.

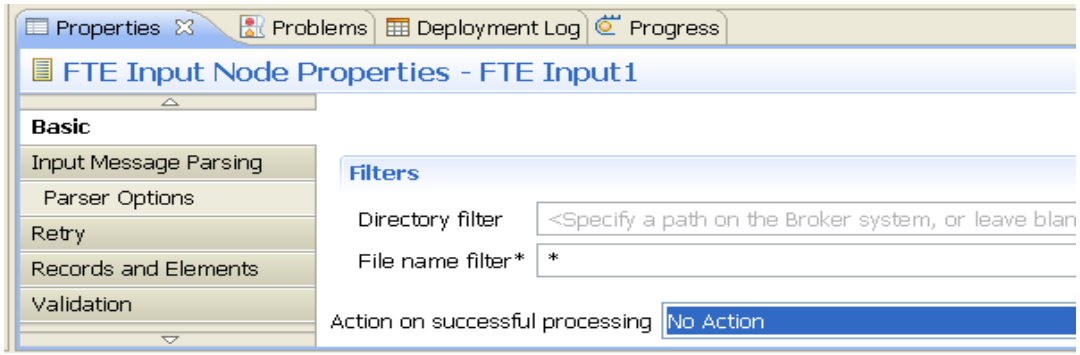
They are new nodes which are based on the current file nodes and have most of the same function where applicable. They have all the record parsing function and the ability to process large files without reading the entire contents into memory at the same time.

FTE input node

FTE Input

FTE Input Node

- Consistent with file input node but makes full use of the power of FTE
- Timely
  - The FTE node is notified by the FTE agent when an inbound transfer is complete
  - The node processes the files in the transfer immediately.
  - Each file is processed independently
  - Can leave file unchanged after processing and just delete notification message
- Metadata
  - Metadata associated with the transfer is sent with the notification
  - Includes user defined metadata
- Filter
  - The node by default receives all transfers
  - Can specify which files to receive using a filter



Notes : FTE Input Node

The FTEInput node has all the core function of the standard File input node but has been enhanced to make use of the powerful function provided by FTE.

It does not require any polling mechanism to scan directories because it is directly trigger by the embedded agent when a file has arrived and the transfer is complete.

It processes each file in a transfer separately and can process each file in parallel.

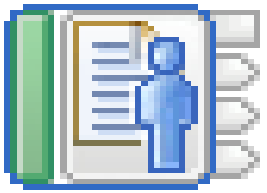
As well as receiving the data from the transfer it also receives all the meta data associated with the transfer. This includes lots of information from FTE but also can include user defined data.

Each FTE Input node can specify a filter of which files it wants to process. By default it processes all files. If two nodes both have a filter which matches a file then only one will get given it to process. By default, if no filter expression is given, then the node we accept any transferred file.

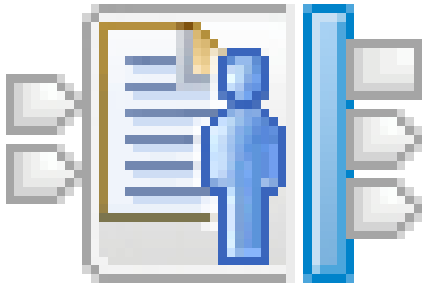
The file name filter accepts wild cards but the directory can either be blank (accept any files) or a string that must match exactly. Relative paths are allowed which are taken relative to the default transfer directory.

FTE Input Node – metadata

LocalEnvironment	
FTE	
Transfer	
Directory	C:\\temp\\perf1
JobName	
Name	test.txt
LastModified	2010-07-05 13:37:55.796
SourceAgent	MB7BROKER.FTESAMPLEIN
DestinationAgent	MB7BROKER.FTESAMPLEIN
OriginatingHost	jreeve6
TransferId	414d51204d4237514d4752202020202045c1314c2000780c
MQMDUser	SYSTEM
OriginatingUser	SYSTEM
TransferMode	binary
TransferStatus	0
FileSize	364
ChecksumMethod	MD5
Checksum	b0b492e8f7385747f3335276615ec5c0
DestinationAgentQmgr	MB7QMGR
SourceAgentQmgr	MB7QMGR
OverallTransferStatus	0
TotalTransfers	1
TransferNumber	1
UserDefined	
com.ibm.jreeve.Test1	Value 1
SourceAgent	MB7BROKER.FTESAMPLEIN
TimeStamp	20100705_123755_796108
Offset	0
Record	1
Delimiter	
IsEmpty	false



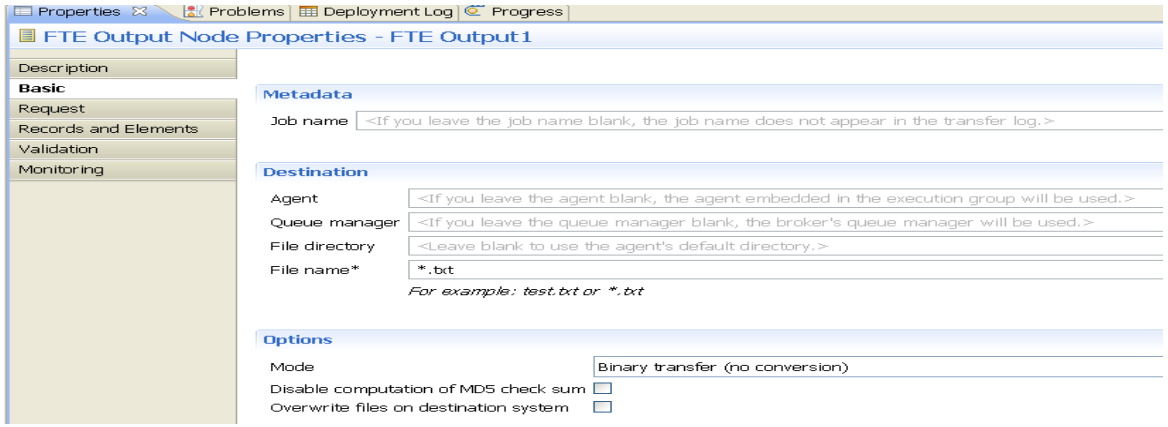
FTE Output node

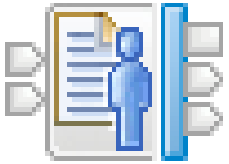


FTE Output

FTE Output Node

- Consistent with file output node but makes full use of the power of FTE
- Transfer details
  - The destination agent, directory etc are defined on the node





  - All details can be overridden using Local environment
  - Support also for wild card file names
- Staging
  - Uses a local staging directory to build up a file record by record for transfer
  - Once a file is finished a request is sent to the FTE Agent to transfer the file
- Metadata
  - User data – if provided in the Local Environment - is sent with the transfer
  - Other metadata is generated by FTE Agent or by Broker
  - Ant scripts – give details of ant scripts to run on remote agent

Notes : FTE Output Node

Again the FTEOuput node makes use of the core function provided by File nodes.

The main properties on the node are details of where to transfer the files to. These can all be overridden using the local environment.

The main difference with an FTEOutput node is that the destination the file is to be written to is not local to the brokers file system but instead is a file system on a remote agent. The node first writes it to a staging directory on the local file system and then sends a request to the embedded agent to transfer it.

Files are built up in a mqsitransmit directory just like with a file node before being moved to the final file name once the file is complete and the transfer request is sent to the FTE agent. The name in the staging area is the same as the name the file will be transferred to unless a file with that name already exists on the local file system. If it exists than a number is appended to the file. The file name it is transferred to on the remote system is not effected by this and will not have the number appended.

FTE Output Node – Overrides and metadata

Overrides:

LocalEnvironment	
Destination	
FTE	
Directory	c:\\temp\\perf1
Name	test.txt
DestinationQmgr	MB7QMGR
DestinationAgent	MB7BROKER.FTESAMPLEIN
JobName	Job 1
Overwrite	true
UserDefined	
com.ibm.jreeve.Test1	Value 1
PreDestinationCall	
Name	john1.xml
PostDestinationCall	
Name	john2.xml

Metadata:

WrittenDestination	
FTE	
Name	test.txt
Directory	c:\\temp\\perf1
DestinationAgent	MB7BROKER.FTESAMPLEIN
DestinationQmgr	MB7QMGR
JobName	Job 1
Overwrite	true
TransferId	414D51204D4237514D4752202020202045C1314C2000780B
UserDefined	
PreDestinationCall	
Name	john1.xml
PostDestinationCall	
Name	john2.xml

Agenda

- Integration using files
- File nodes
- FTE nodes
- CD nodes
- Additional SupportPac nodes
  - VSAM nodes
  - QSAM nodes

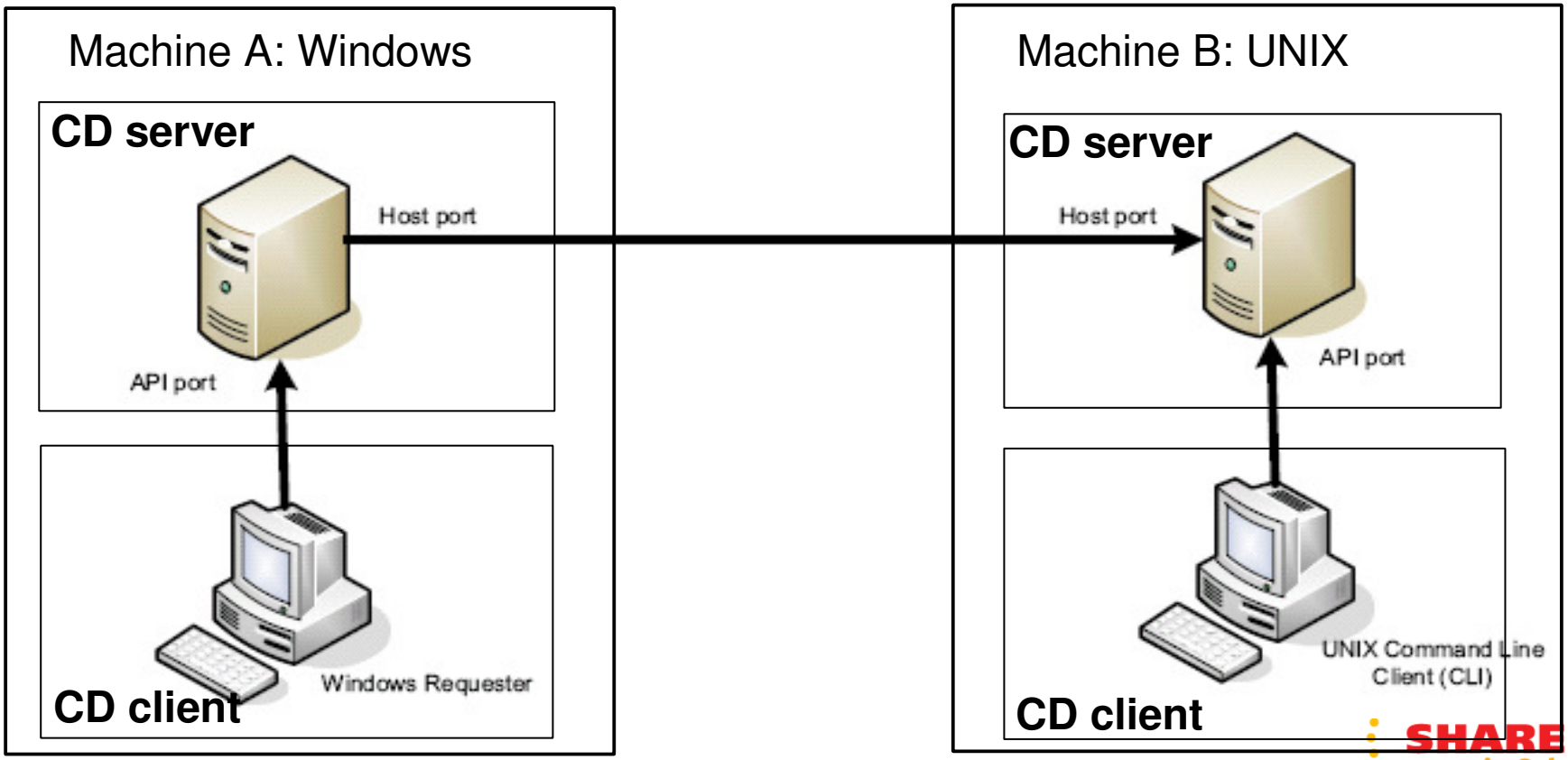
CD nodes in new function APAR IC75621

CD Input

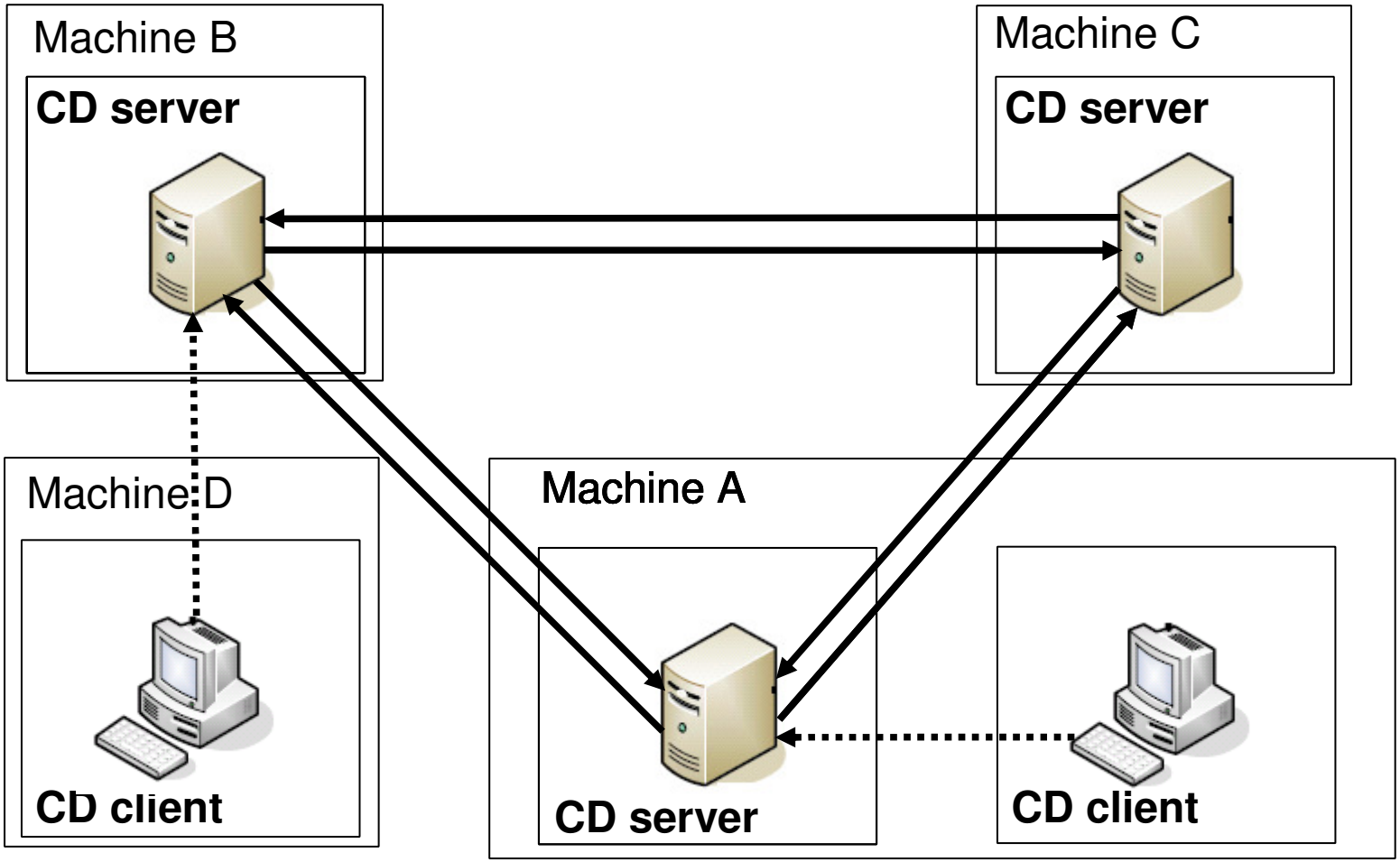
CD Output

IBM Sterling Connect:Direct (no Message broker)

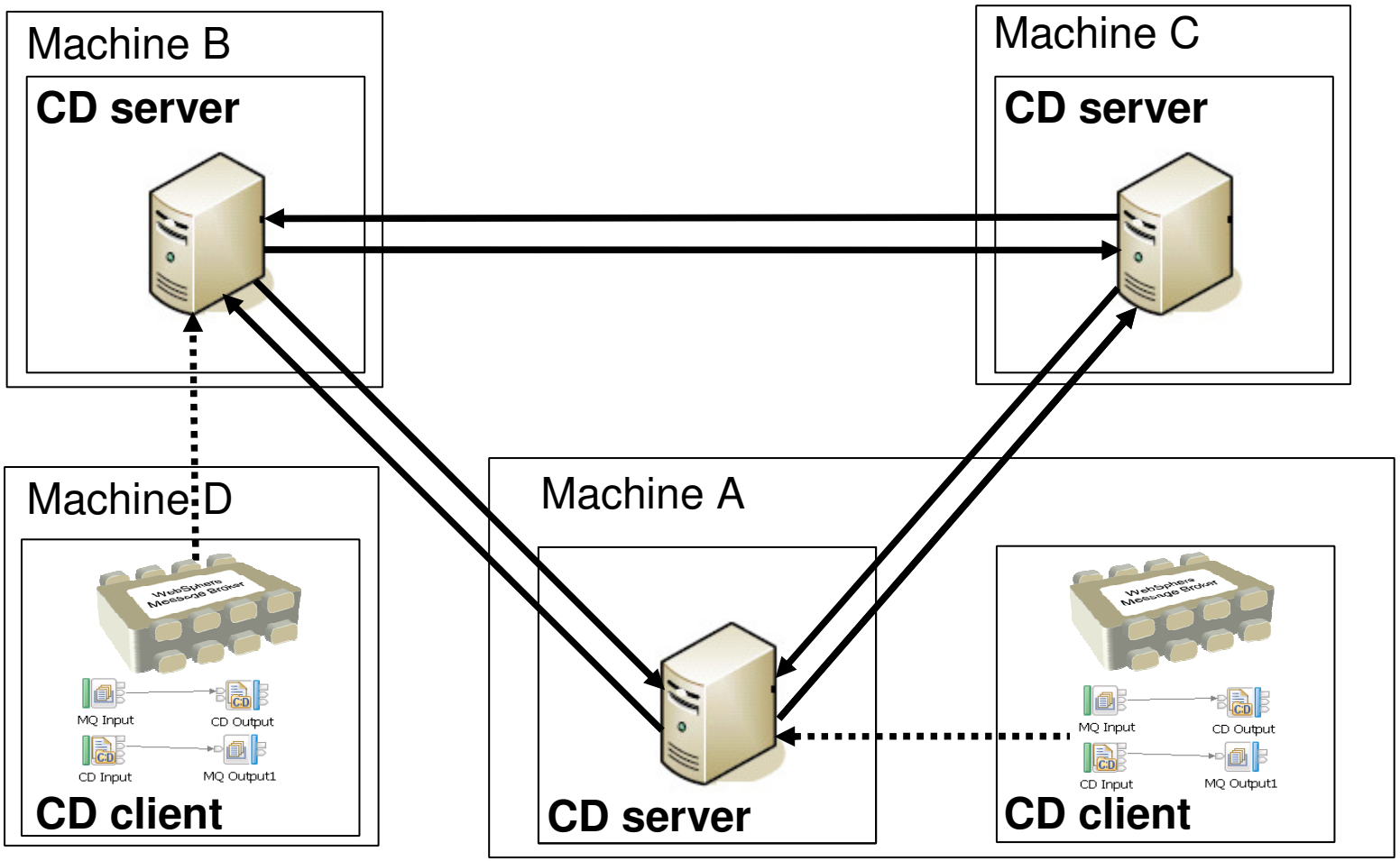
- IBM Sterling Connect:Direct is a managed file transfer solution



IBM Sterling Connect:Direct (no Message Broker)

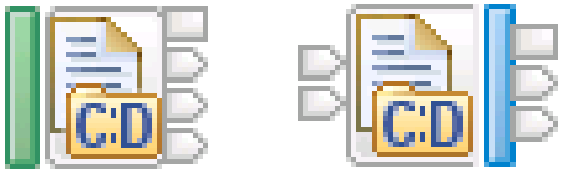


IBM Sterling Connect:Direct (with Message Broker)



Adding WMB to a CD network

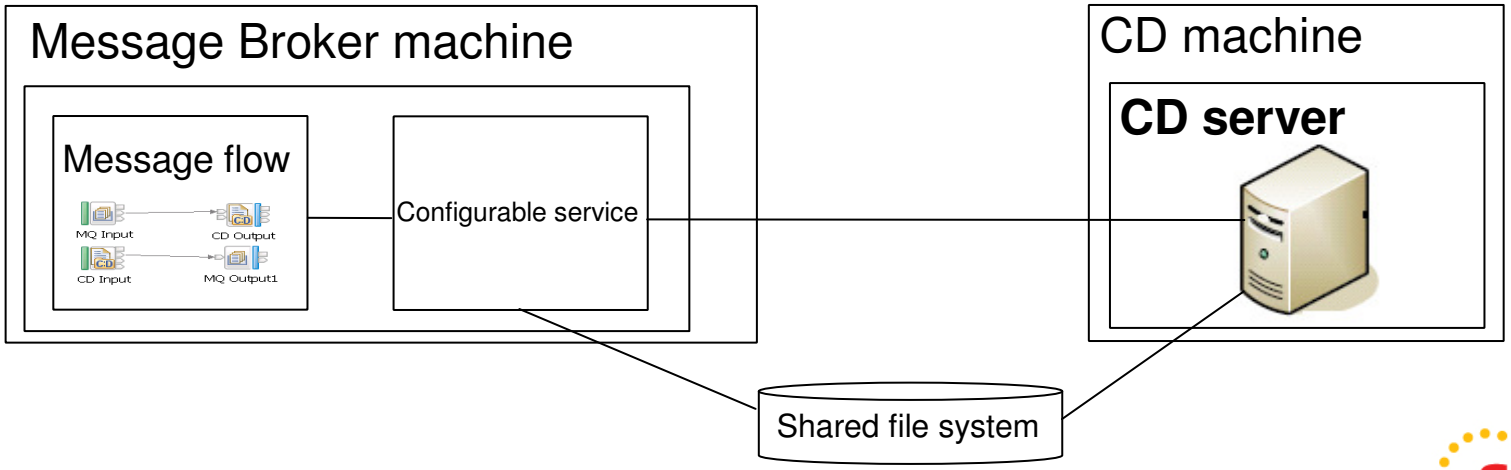
- Install WebSphere Message Broker
- Create and start a broker on the machine
- Set up a security identity to be used to connect to CD server:
  - mqsisetdbparms BROKER –n cd::default –u jreeve –p ********
- Create and deploy flows which interact with CD via built in CD nodes



- That is all that is needed if the broker is on the same machine

Adding WMB to a CD network

- Message Broker does not have to be on the same machine as CD server.
- They must have access to a shared file system where files are transfer to.
- A configurable service can be created with details of CD Server:
  - Hostname
  - API port to connect to
  - Security identity
  - Filepath of the shared file systems



Accessing a CD server from a message flow

- Messages sent and received from the CD server using Message Broker nodes:
  - CDInput node: receives any file transferred to the CD server
  - CDOutput node: constructs a file and sends a request to the CD server to transfer the file to a remote system.
- New CD nodes but based on the current file nodes allowing:
  - Record based processing
  - Stream parsing for large file support
  - Can be used with all other Broker nodes
- CD server is not stopped, started or administered by Message Broker. Both products are decoupled:
  - Message Broker outages have no effect on CD server
  - CD server outages only effect Message Broker when a transfer is requested by a message flow

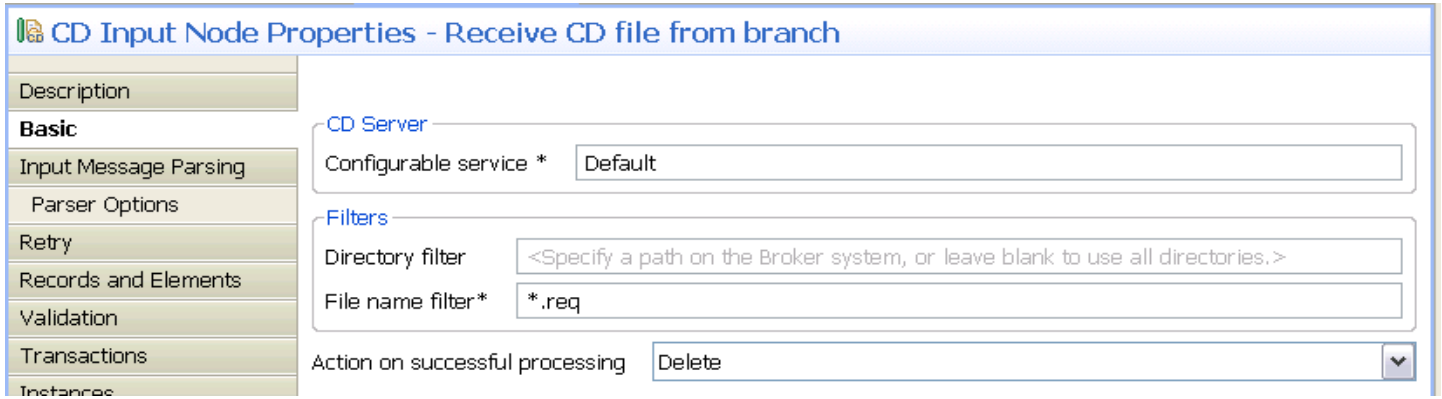
CD Input Node

- Consistent with file input node but makes full use of CD function

- Timely
  - The CD node monitors the CD servers stats for transfers that have completed
  - The node processes the files in the transfer immediately.
  - Each copied file is processed independently
  - Can leave the file unchanged after processing and just delete notification message

- Metadata
  - Metadata associated with the transfer is sent with the notification
  - Includes user defined application data field

- Filter
  - The node by default receives all transfers
  - Can specify which files to receive using a filter

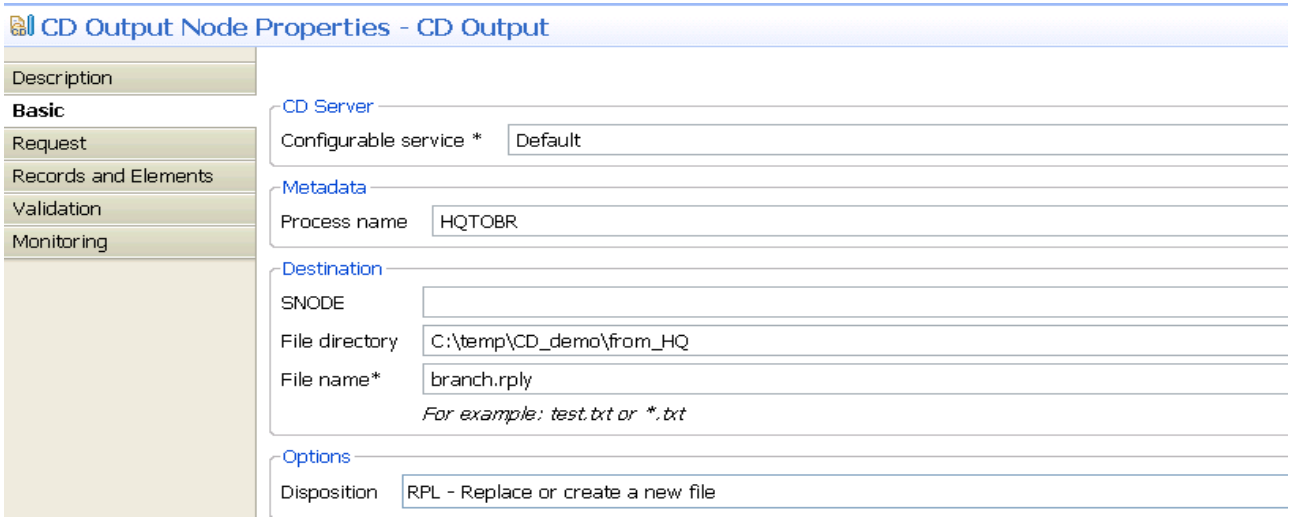


CD Input Node – metadata

LocalEnvironment	
CD	
Transfer	
ProcessName	BRTOHQ
Step	CPBRHQ
ProcessNumber	20536
Submitter	mqbroker
Accounting	
SourcePath	C:\\temp\\CD_demo\\to_HQ\\Account_transfers.req
DestinationPath	C:\\Program Files\\Sterling Commerce\\Connect Direct v
PNODE	JREEVE6
SNODE	JREEVE6
TimeStamp	20110303_174443_333773
Offset	0
Record	1
Delimiter	
IsEmpty	false

CD Output Node

- Consistent with file output node but makes full use of CD function
- Transfer details
  - The destination CD server, directory etc are defined on the node



  - Support also for wild card file names
- Staging
  - Uses a local staging directory to build up a file record by record for transfer
  - Once a file is finished a request is sent to the CD Server to transfer the file
- Metadata
  - User data – if provided in the Local Environment - is sent with the transfer
  - Other metadata is generated by CD server or by Broker

CD Output Node – Metadata

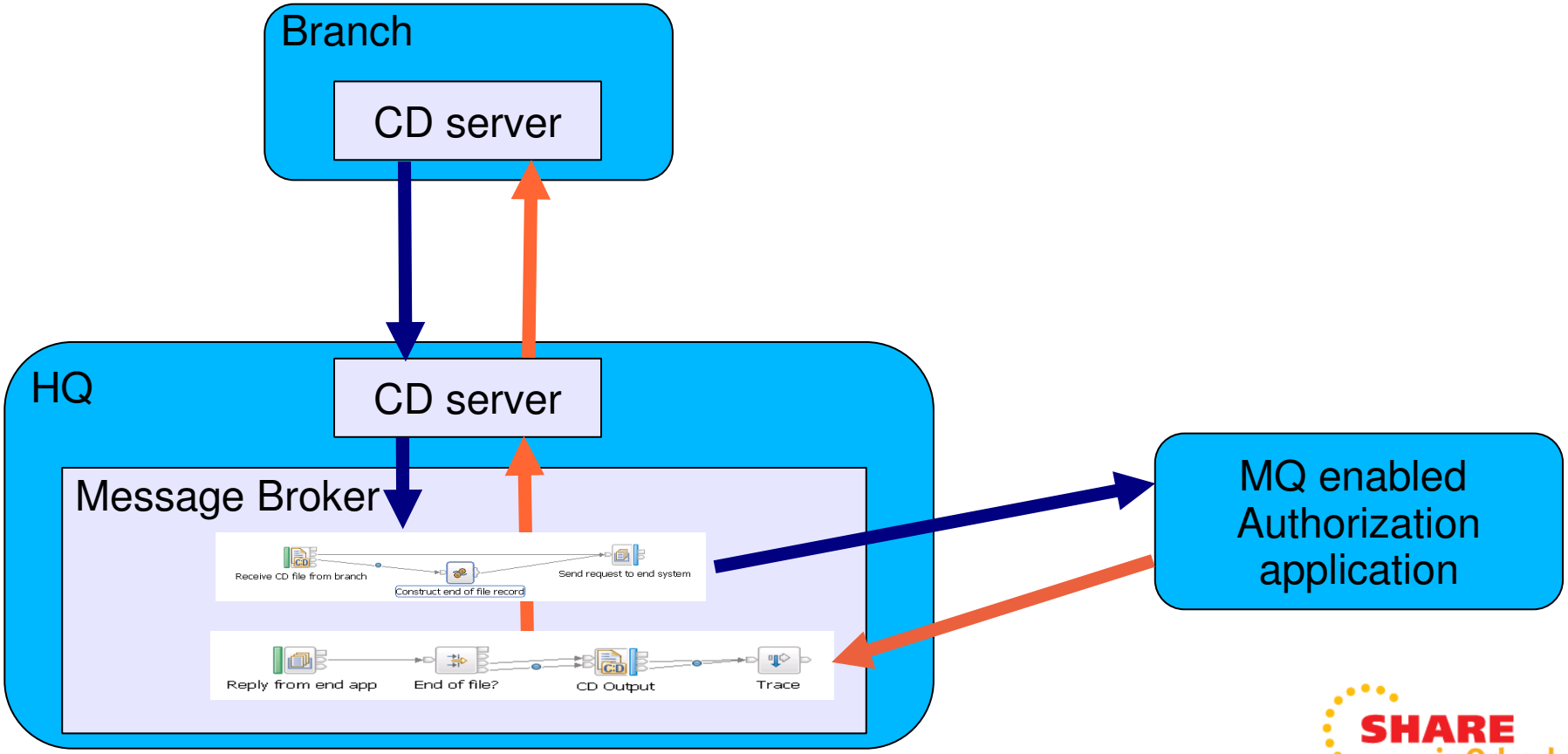
LocalEnvironment	
WrittenDestination	
CD	
ProcessName	HQTOBR
ProcessNumber	20542
Directory	C:\\temp\\CD_demo\\from_HQ
Name	branch.rply
PrimaryNodeName	JREEVE6
PrimaryNodeOS	Windows
SecondaryNodeName	JREEVE6
SecondaryNodeOS	Windows

File types supported

- Flat files on windows, UNIX and z/OS
  - Treated and processed the same as in the normal file nodes
- z/OS Sequential datasets
  - File name is the dataset name: JREEVE.TEST1.TEST2
  - Wildcard values allowed anywhere: JREEVE.*.*
  - Staged to HFS and then treated the same as in the normal file nodes
- z/OS Partitioned datasets
  - For the input node file name is a pattern like:
    - JREEVE.TEST1.TEST3(MEM1)
    - JREEVE.TEST1.TEST3(MEM*)
    - JREEVE.TEST1.TEST3(*)
    - JREEVE.*.TEST3(*)
  - Each member is staged to HFS and then treated the same as in the normal file nodes
  - For the output node the file name must be a single member:
    - JREEVE.TEST1.TEST3(MEM1)

Scenario

"A bank has a clearing system for transfers between accounts where each bank branch collects important transfers between accounts during the day in a batch file. The transfers are sent to a central HQ for authorization during the night. The HQ then sends a batch file back to the branch containing status of the authorizations on each transfer"



Agenda

- Integration using files
- File nodes
- FTE nodes
- CD nodes
- Additional SupportPac nodes
  - VSAM nodes
  - QSAM nodes

SupportPac IA13 – VSAM nodes

- A suite of nodes allowing users to perform record oriented processing on VSAM files.
  - VSAMDelete
  - VSAMInput
  - VSAMRead
  - VSAMUpdate
  - VSAMWrite

- The broker can process VSAM records in the same way as it processes messages from or to other data sources.
- A suite of 5 nodes allowing users to perform record oriented processing on VSAM files: Input, Read, Write, Update and Delete operations.
- Users can combine these nodes for VSAM access with other nodes to integrate VSAM processing into message flows, or use VSAM files to drive message flow processing
- VSAM file support
  - KSDS, ESDS, RRDS, KSDS_PATH, ESDS_PATH
- **Fully supported Cat 3 SupportPac for V7**

Notes : VSAM Nodes

- **Suite of Five nodes**
  - A comprehensive set of nodes allowing you to perform record oriented processing on VSAM files.
  - You can start a message flow with the VSAM input node, and there are corresponding nodes to perform subsequent read, write, update and delete operations on VSAM files.
  - These nodes can be mixed and matched with all other supplied nodes to create sophisticated VSAM processing capabilities.
- **Input Node**
  - The VSAM Input node is targeted at batch oriented processing. This node can read a set of records from a VSAM file (identified by an input control message) and propagate them down connected nodes.
  - Unlike message oriented processing with MQ, VSAM does not have an obvious **trigger** to identify when a file is to be processed.
  - The “control” terminal provides the trigger. This is usually wired to a different message flow, so that VSAM processing can be driven from another flow’s processing – it provides the trigger.
  - This node also requests additional instances, so that multiple threads can be processing a VSAM file in parallel, if processing allows.
- **Read Node**
  - Whereas the input node can read a set of records, the read node reads a single record, and places it at a user specified location in the message tree.
  - Properties identify and parse the record to be read
    - Basic, Default, Request groups (Key, RBA, RRN, …)
  - Additional Properties
    - Advanced, Result, Status groups
  - VSAM Request message control message overrides some of these properties to allow very flexible and programmatic processing.
- **Write, Update, Delete Nodes**
  - These all follow the same pattern as read node, performing write, update or delete operations respectively.

VSAM Usage Scenarios

1. Batch input processing

- A triggering event will enable the broker to open a VSAM file, read a batch of records and propagate these as input messages.

2. Batch input processing with update

- Similar to 1, but includes optimizations for later updates to the input record.

3. Data enrichment from VSAM

- Data from a VSAM file can be used to modify or route an in-flight message.

1. Data logging to VSAM

- Data taken from an in-flight message can be used to update a VSAM record or to create a new record.

2. Deletion of VSAM data

- Traversing a message flow path can cause specified VSAM records to be deleted.

SHARE in Orlando 2011

Notes : VSAM Usage Scenarios

- **Scenarios**
  - **There are several uses for the VSAM node, but essentially, we want to be able to process VSAM records as we would do messages.**
  - **Moreover, you can combine these nodes for VSAM access with other nodes to integrate VSAM processing into message flows, or use VSAM files to drive message flow processing.**
- **Batch Input Processing**
  - **Use the** VSAMInput **node to read a specified number of records from a VSAM data set and propagate each record to subsequent nodes in your message flow. You can configure the node to read either every record in the data set, or a specified number of records from a part of the data set.**
- **Batch Input Processing with Update**
  - **This is essentially the same as scenario above, but allows (for example) records to be locked for update later in the flow.**
- **Data Enrichment and Routing from VSAM**
  - **Use VSAM as you would use a database to perform data enrichment or message routing.**
- **Data Logging to VSAM**
  - **Record interesting events to a VSAM file.**
- **Deletion of VSAM data**
  - **Remove VSAM file records on the basis of message processing.**

- **Transactionality** The VSAM operations that are performed by these nodes do not participate in the transactions that exist within the message flow. Successful VSAM operations are not automatically backed out if a failure occurs in the message flow. As such you will need to bear this behaviour in mind when designing applications using the VSAM nodes.

- **SHAREOPTIONS**. When you consider the effect of using VSAM SHAREOPTIONS on a VSAM data set, note that an execution group rather than a message flow should be regarded as the equivalent of a user task or application program. **Note:** The services that are used by the VSAM nodes to access VSAM data sets do not use SHAREOPTIONS to enforce which intents the data set can be opened with. Therefore, any VSAM data set can be opened for read or update intent by the VSAM nodes, regardless of the setting of SHAREOPTIONS.

SupportPac IA11 – QSAM nodes

- These are similar in concept and usage to the VSAM nodes, but oriented around sequential files, rather than record oriented files.
  - FileDelete
  - FileRead
  - FileRename
  - FileWrite

- XML action control messages used to indicate when to open, close, delete rename files.
- Facilities provided
  - File to file
  - File to queue
  - Queue to file
- **Fully supported Cat 3 SupportPac for V7**

Summary

- Comprehensive file processing support in Message Broker
  - File
  - Large file and Record processing support
  - FTP
  - SFTP
  - QSAM and VSAM support

- From 7.0.0.1 file transfer function has been added using an embedded WMQ FTE agent
- From 7.0.0.2 plus APAR IC75621 support for IBM Sterling Connect:Direct
- Files are a key focus area for future releases

The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00			More than a buzzword: Extending the reach of your MQ messaging with Web 2.0	Batch, local, remote, and traditional MVS - file processing in Message Broker	Lyn's Story Time - Avoiding the MQ Problems Others have Hit
09:30		WebSphere MQ 101: Introduction to the world's leading messaging provider	The Do's and Don'ts of Queue Manager Performance	So, what else can I do? - MQ API beyond the basics	MQ Project Planning Session
11:00		MQ Publish/Subscribe	The Do's and Don'ts of Message Broker Performance	Diagnosing problems for Message Broker	What's new for the MQ Family and Message Broker
12:15	MQ Freebies! Top 5 SupportPacs	The doctor is in. Hands-on lab and lots of help with the MQ family		Using the WMQ V7 Verbs in CICS Programs	
01:30	Diagnosing problems for MQ	WebSphere Message Broker 101: The Swiss army knife for application integration	The Dark Side of Monitoring MQ - SMF 115 and 116 record reading and interpretation	Getting your MQ JMS applications running, with or without WAS	
03:00	Keeping your eye on it all - Queue Manager Monitoring & Auditing	The MQ API for dummies - the basics	Under the hood of Message Broker on z/OS - WLM, SMF and more	Message Broker Patterns - Generate applications in an instant	
04:30	Message Broker administration for dummies	All About WebSphere MQ File Transfer Edition	For your eyes only - WebSphere MQ Advanced Message Security	Keeping your MQ service up and running - Queue Manager clustering	
06:00			Free MQ! - MQ Clients and what you can do with them	MQ Q-Box - Open Microphone to ask the experts questions	