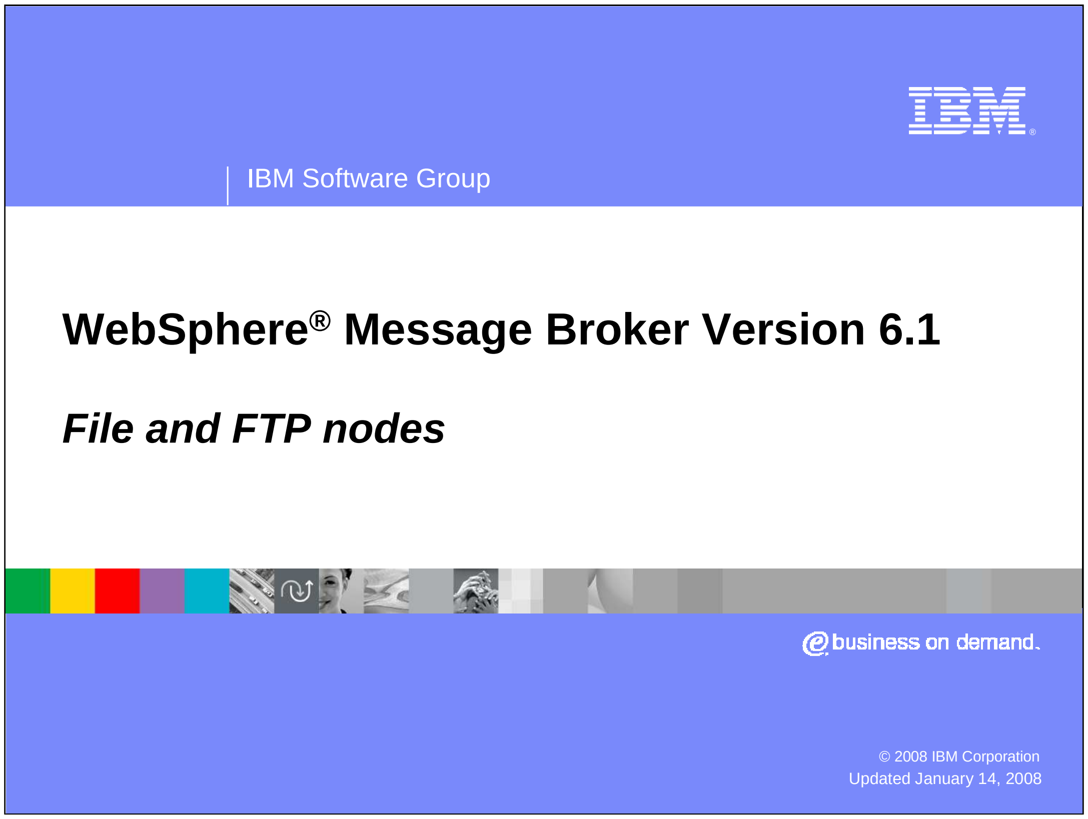
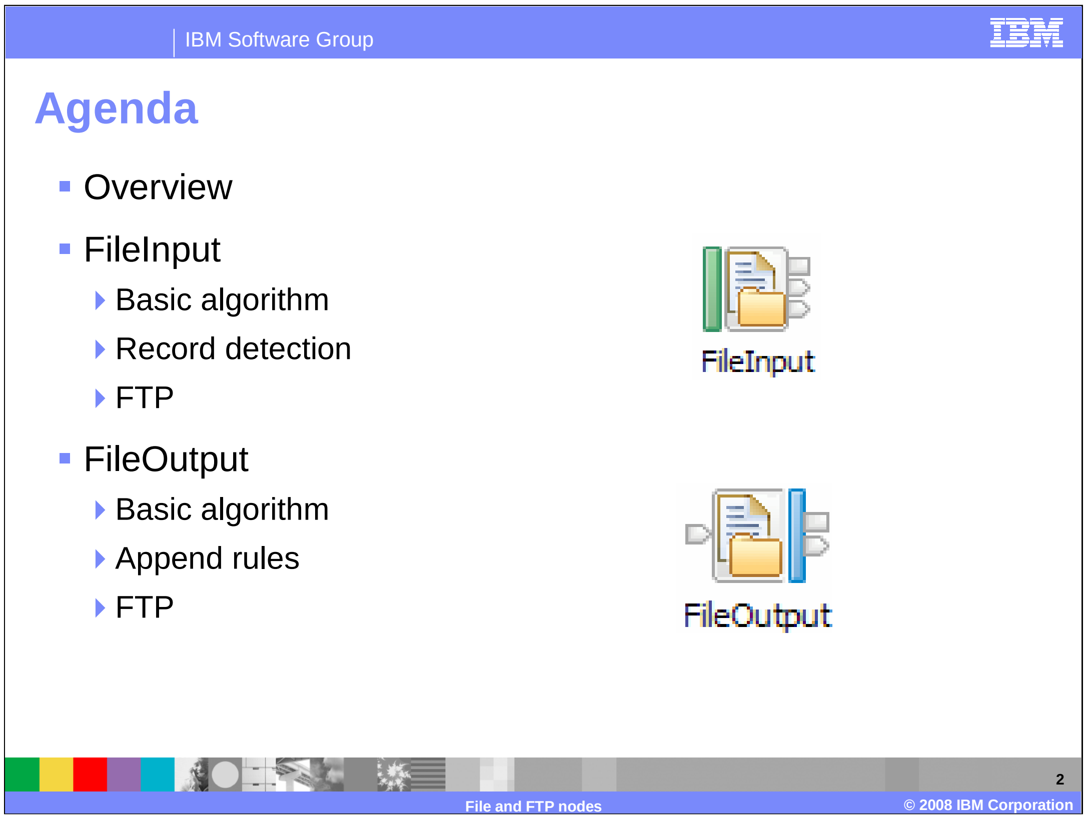


IBM Software Group

WebSphere® Message Broker Version 6.1

File and FTP nodes

This presentation will discuss the new File functions in WebSphere Message Broker Version 6.1.

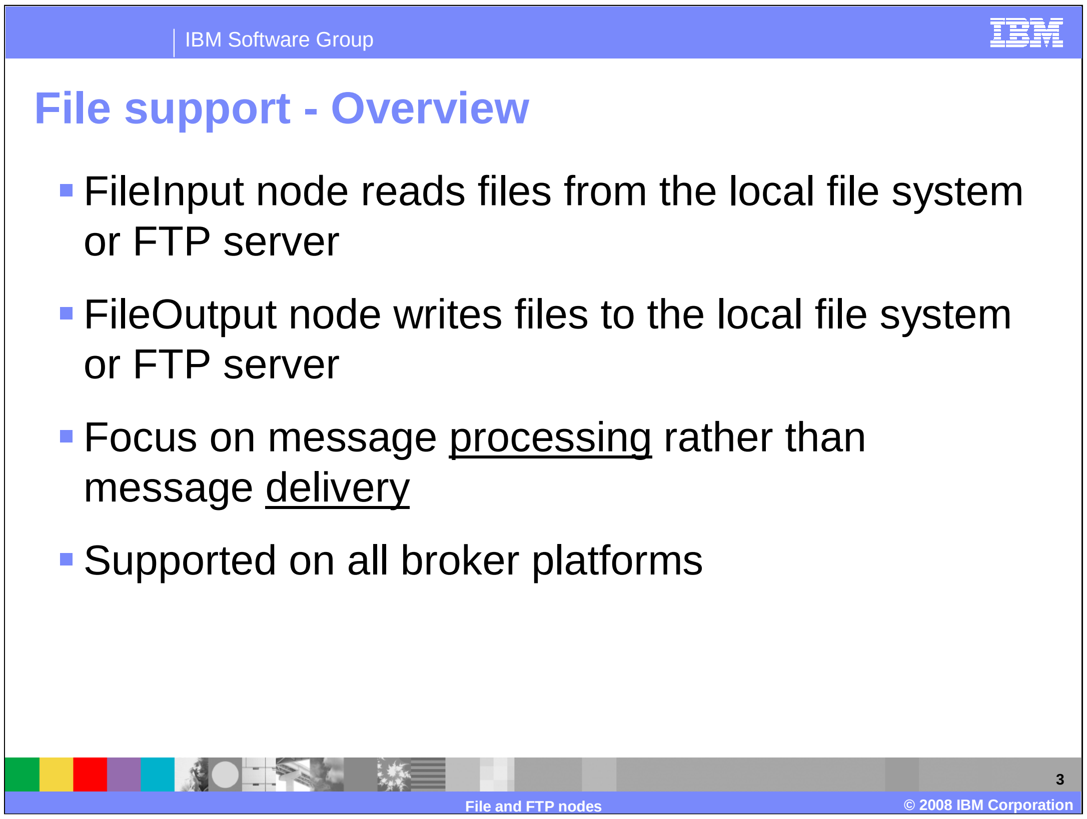


Agenda

- Overview
- FileInput
  - Basic algorithm
  - Record detection
  - FTP
- FileOutput
  - Basic algorithm
  - Append rules
  - FTP

FileInput
FileOutput

After an overview of the File support added in Version 6.1, this presentation will discuss in greater detail the two new nodes that provide this capability – File Input and File Output. For both nodes, the presentation will first cover the basic algorithm, then discuss how the nodes provide support for multiple records in a single file. Finally for each node, it will cover the enhancements that allow the node to communicate with FTP servers.



File support - Overview

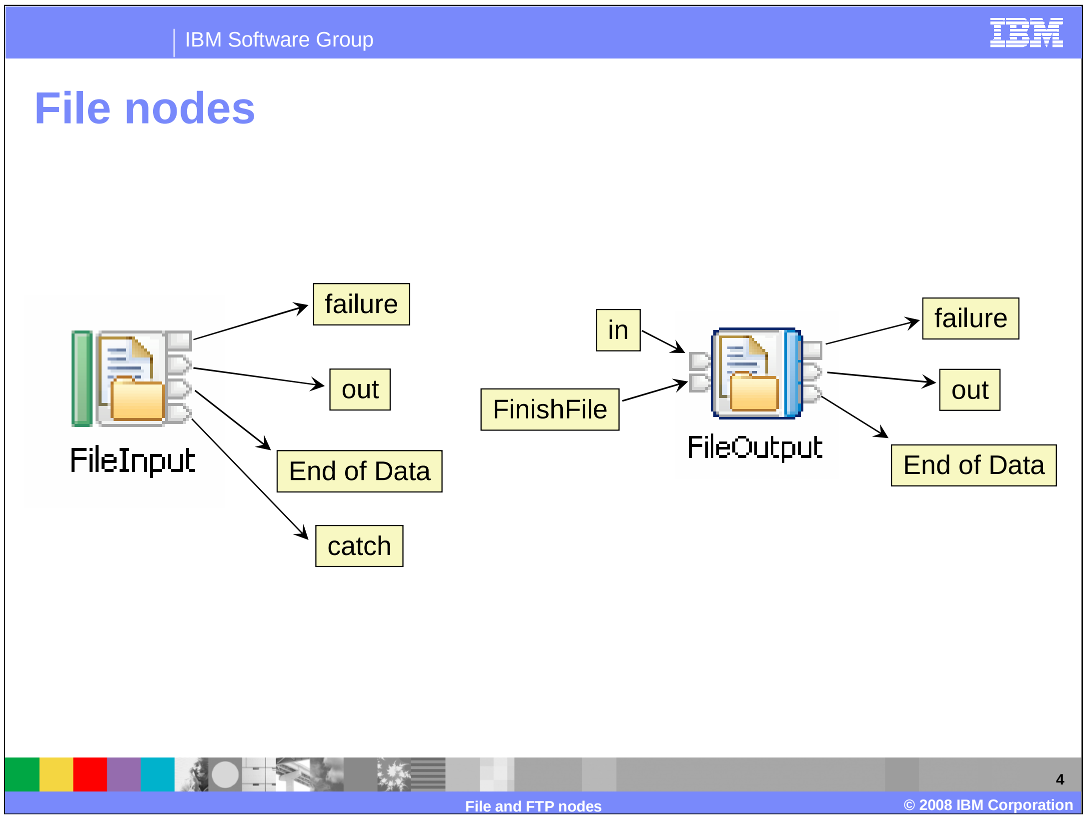
- FileInput node reads files from the local file system or FTP server
- FileOutput node writes files to the local file system or FTP server
- Focus on message <u>processing</u> rather than message <u>delivery</u>
- Supported on all broker platforms

Support has been added in Message Broker version 6.1 for both inbound and outbound file manipulation. It is possible for the broker to read files from the local file system and write files back to the file system. It is also possible to read and write files to or from an FTP server.

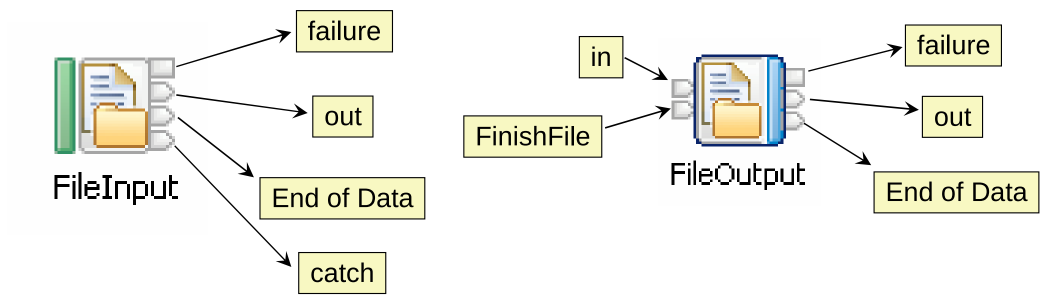
The focus of the file support in Message Broker is on message processing rather than message delivery. To move files reliably, a product such as WebSphere PM4Data is recommended.

Support for the File Nodes is available on all platforms. On z/OS, files will be resolved in the HFS or ZFS file system, and file names treated in the case-sensitive UNIX manner.

While this is the first time that file support has been delivered as part of the core product offering, it is not the first file capability to be available in the Message Broker. For example, support for VSAM files on the z/OS platform has been possible for a while through a category 3 SupportPac. Additionally, the Message Broker File Extender product is available for Message Broker version 6.0. The file support discussed in this presentation is not based on or connected with this product extension.


IBM Software Group

File nodes



4

File and FTP nodes

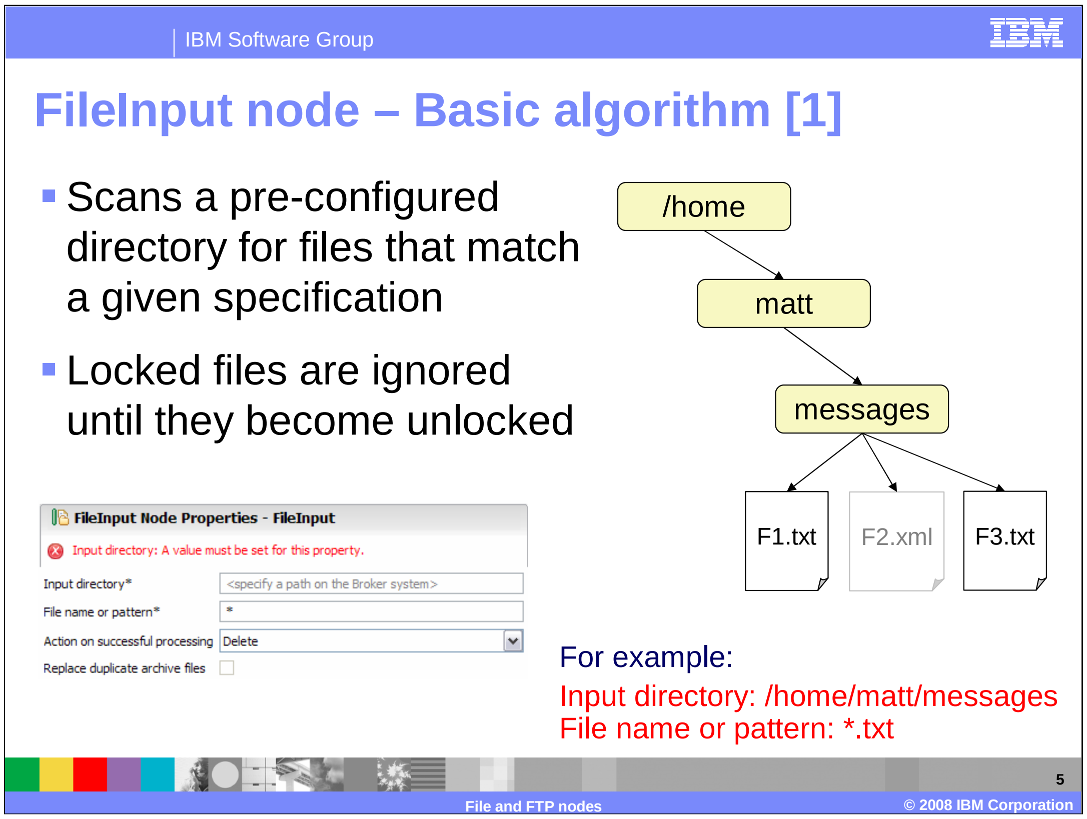
© 2008 IBM Corporation

Here are the two nodes that provide file support in Message Broker version 6.1. Both nodes have an out terminal (used after successful processing), and a failure terminal (used after processing error occurs in the node). As an input node, File Input also has a catch terminal that is used to flow caught errors that occur downstream of the node.

There are specific terminals, both on the File Input and File Output Nodes, for handling the end of the file.

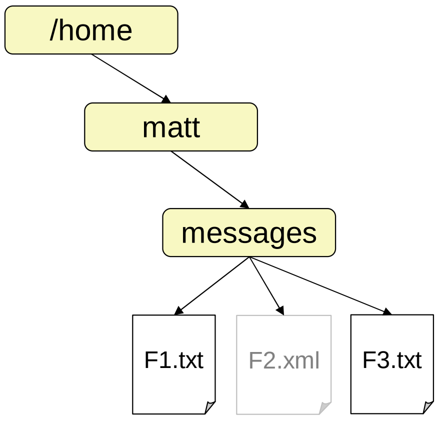
On the File Input Node, the “End of Data” terminal is given control when the Broker detects that it has reached the end of the input file.

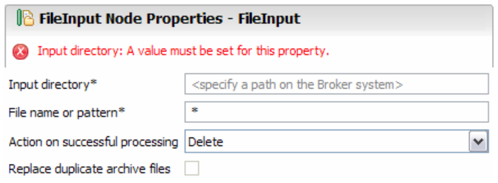
On the Output Node, the “Finish File” terminal is driven to indicate that the file should be written out. Assuming this is completed successfully, the message sent to the “Finish File” terminal is propagated to the “End of Data” terminal.



FileInput node – Basic algorithm [1]

- Scans a pre-configured directory for files that match a given specification
- Locked files are ignored until they become unlocked





FileInput Node Properties - FileInput

Input directory: A value must be set for this property.

Input directory*	<specify a path on the Broker system>
File name or pattern*	*
Action on successful processing	Delete
Replace duplicate archive files	☐

For example:
Input directory: /home/matt/messages
File name or pattern: *.txt

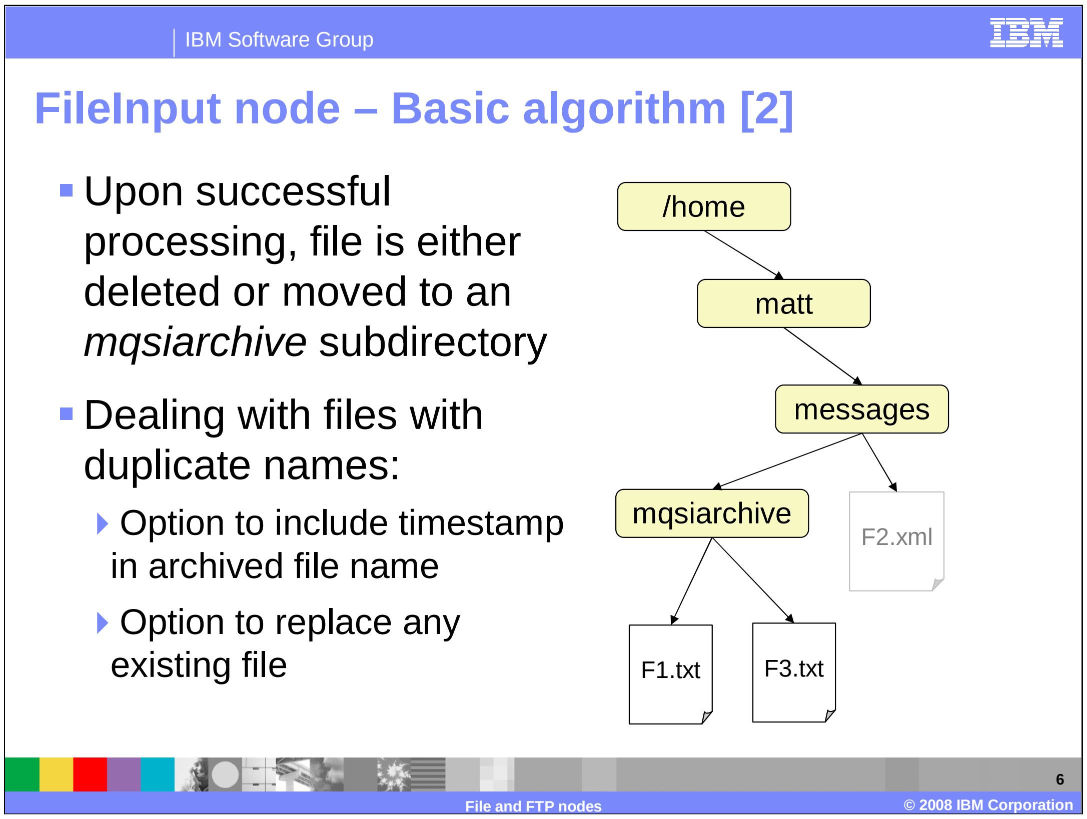
The next few slides describe the basic algorithm for file processing.

The File Input node is able to read through a pre-configured directory on the broker's file system. The directory name is configured on the Basic tab of the node and must be entered in the format expected by the broker file system. This directory will be scanned by the Broker for files. Subdirectories of this directory are not scanned. The files to be processed must be present in the specified directory.

The file name can be specified as a wildcard. Acceptable wildcards on the file name pattern are 'asterisk*' which matches multiple characters, and 'question mark', which matches a single character.

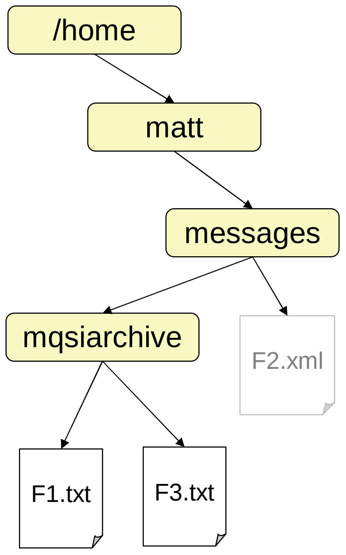
The value for the directory must either be a fully qualified directory path, or as a directory relative to $MQSI_WORKPATH/file. For example, in a Windows® environment the default is C:\ Documents and Settings \ All Users \ Application Data \ IBM \ MQSI \ file.

If there is an operating system lock on a file it will not be read until the lock is removed.



FileInput node – Basic algorithm [2]

- 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 file name
  - Option to replace any existing file



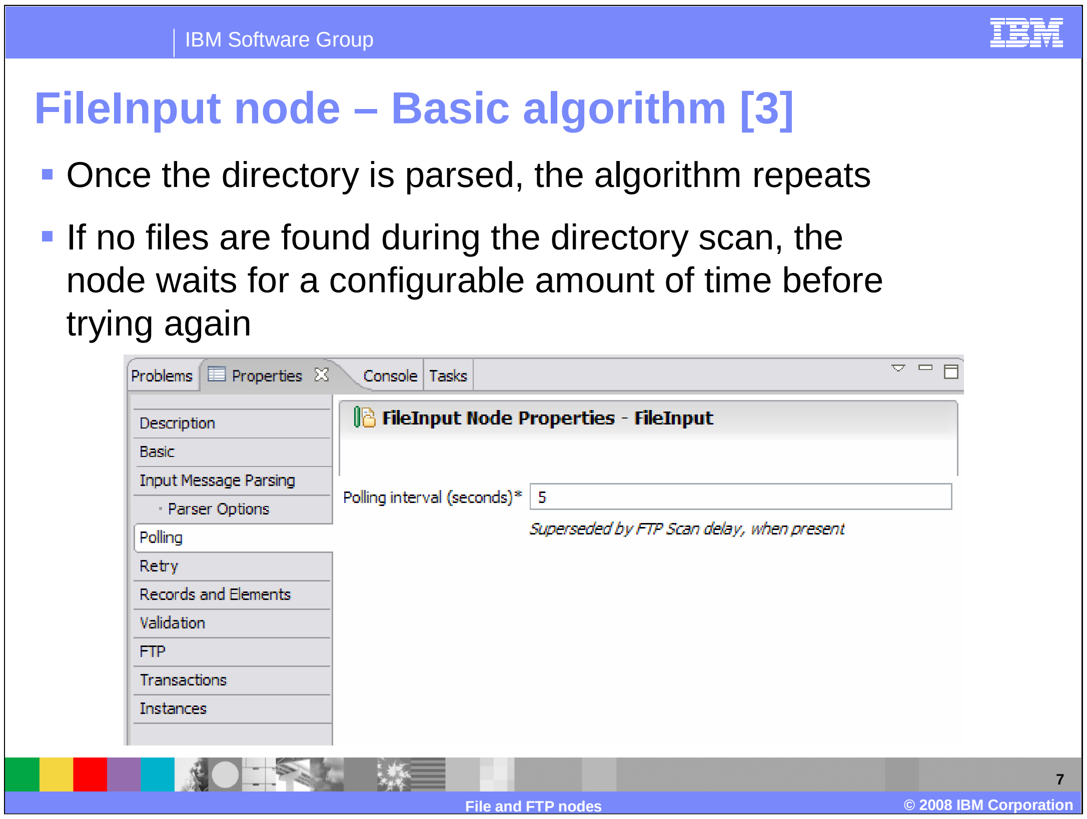
Once the file is read and parsed, an option on the node specifies the action to take; either the input file is deleted, or moved into an archive subdirectory of the input directory. If the archive directory does not exist, it will be created.

The archive directory will be named “mqsiarchive”

If an archive file already exists with the same name, an option exists on the node to overwrite it. If you do not set this option, the node will throw an exception if it tries to move a successfully-processed file with the same name into the archive sub-directory.

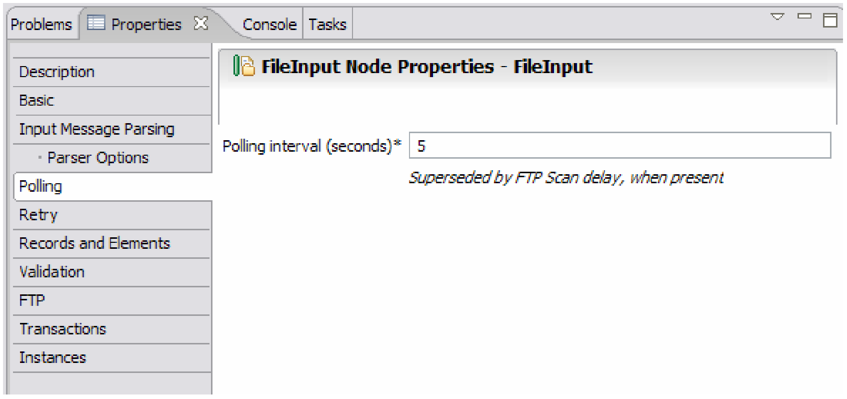
Alternatively, you can opt to rename the file to include a timestamp value as it is moved into the archive directory, which will prevent duplicate files in the directory. The timestamp includes the time to a millisecond granularity, ensuring uniqueness of file names in the archive directory.

When the file is placed into the archive directory, the contents of the file will not be altered.



FileInput node – Basic algorithm [3]

- Once the directory is parsed, the algorithm repeats
- If no files are found during the directory scan, the node waits for a configurable amount of time before trying again



FileInput Node Properties - FileInput

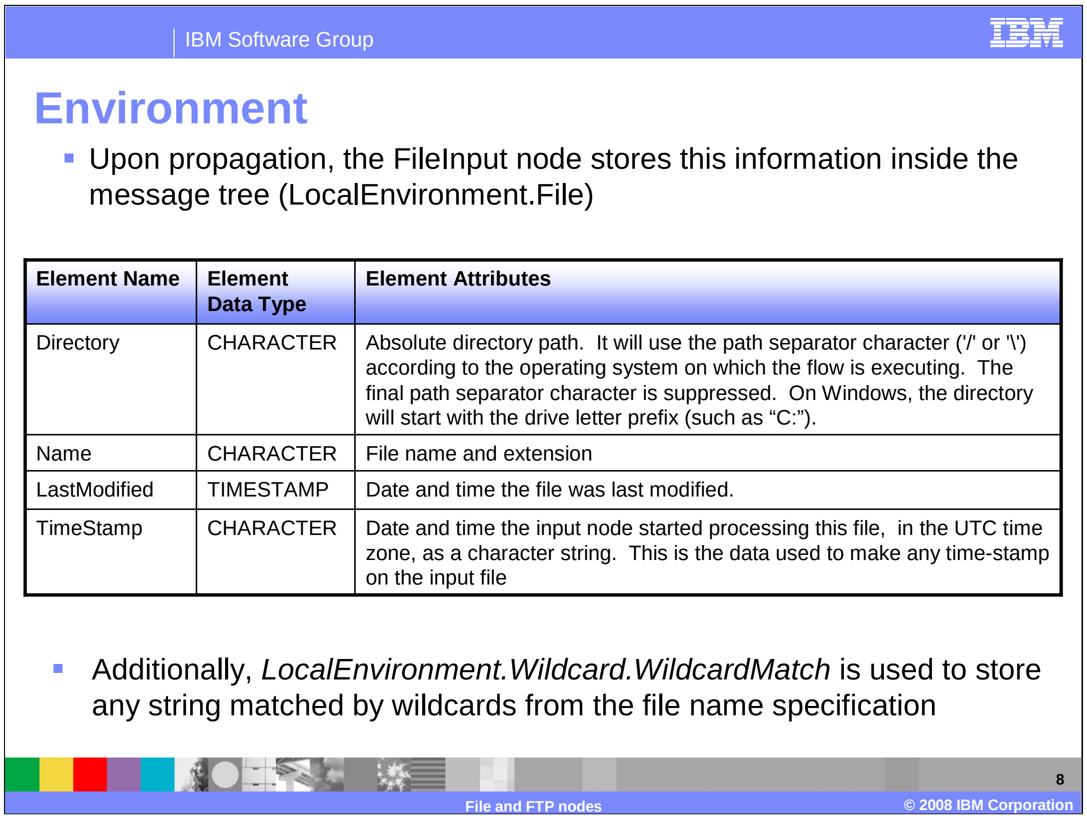
Polling interval (seconds)* 5

Superseded by FTP Scan delay, when present

Once the file is processed, the next file in the directory is processed.

Once all files have been processed, the directory is rescanned. If any files that were previously locked are now unlocked, they will be processed by the Broker.

If a directory scan yields no matches, the node waits for a preconfigured amount of time before rescanning. The default value for this interval is 5 seconds.


IBM Software Group

Environment

- Upon propagation, the FileInput node stores this information inside the message tree (LocalEnvironment.File)

Element Name	Element Data Type	Element Attributes
Directory	CHARACTER	Absolute directory path. It will use the path separator character ('/' or '\\') according to the operating system on which the flow is executing. The final path separator character is suppressed. On Windows, the directory will start with the drive letter prefix (such as “C:”).
Name	CHARACTER	File name and extension
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time the input node started processing this file, in the UTC time zone, as a character string. This is the data used to make any time-stamp on the input file

- Additionally, *LocalEnvironment.Wildcard.WildcardMatch* is used to store any string matched by wildcards from the file name specification

8

File and FTP nodes

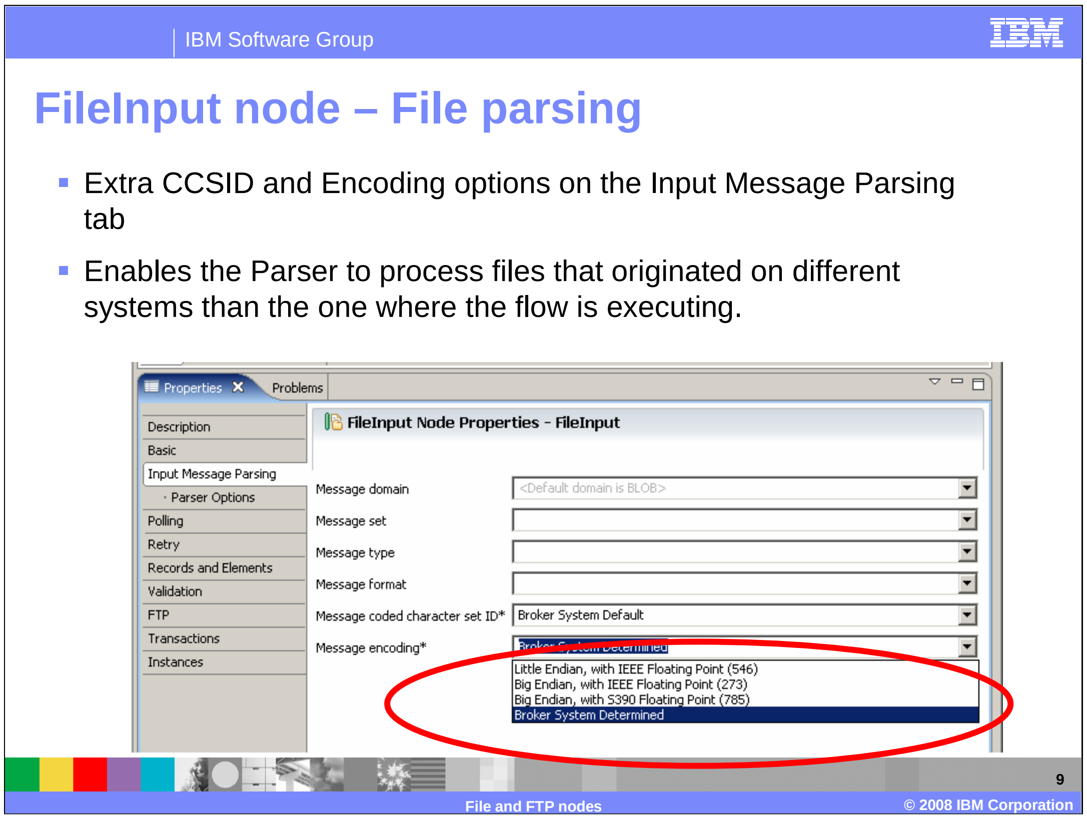
© 2008 IBM Corporation

Each time the contents of a file are propagated, the File Input node stores additional information on the file inside the Local Environment, and this table describes those fields.

For example, the Local Environment Directory element contains the absolute directory name that contains the file that is being processed.

Other element values are as shown in the table on this slide.

Also, the File Input node copies the characters in the file name matched by wild cards, together with any intermediate characters, to the Properties tree “*WildcardMatch”* element. So if the match expression was “file*.txt” and the file read was “file02.txt”, then the value of WildcardMatch would be “02”. Or if the match expression was “file??type.*”, and the file read was “file02type.xml”, the value of WildCardMatch would be “02type.xml”. This information can be used by an output node, where the output name can be derived from the input name.



FileInput node – File parsing

- Extra CCSID and Encoding options on the Input Message Parsing tab
- Enables the Parser to process files that originated on different systems than the one where the flow is executing.

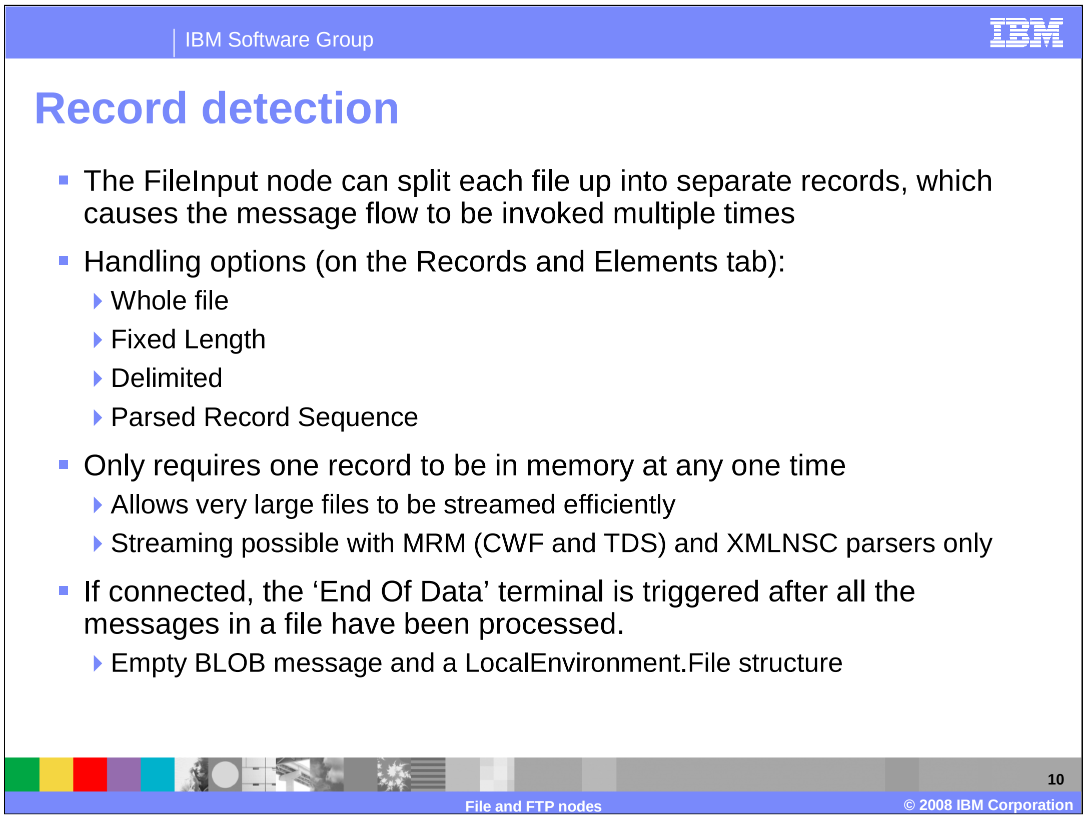
FileInput Node Properties - FileInput

Message domain	<Default domain is BLOB>
Message set	
Message type	
Message format	
Message coded character set ID*	Broker System Default
Message encoding*	Broker System Determined

Little Endian, with IEEE Floating Point (546)
Big Endian, with IEEE Floating Point (273)
Big Endian, with S390 Floating Point (785)
Broker System Determined

Before covering how the file information is parsed, note that the Input Message Parsing tab on the File Input Node has two non-standard fields (CCSID and encoding). This is similar to the equivalent properties on an MQ Input Node.

The File Input node reads raw bytes from the file system before passing on to the parser. These parameters make it possible to parse information from a system other than the one in which the broker is running. This system may have a different Character Set than the one hosting the Broker, and this property allows these files to be processed correctly.



Record detection

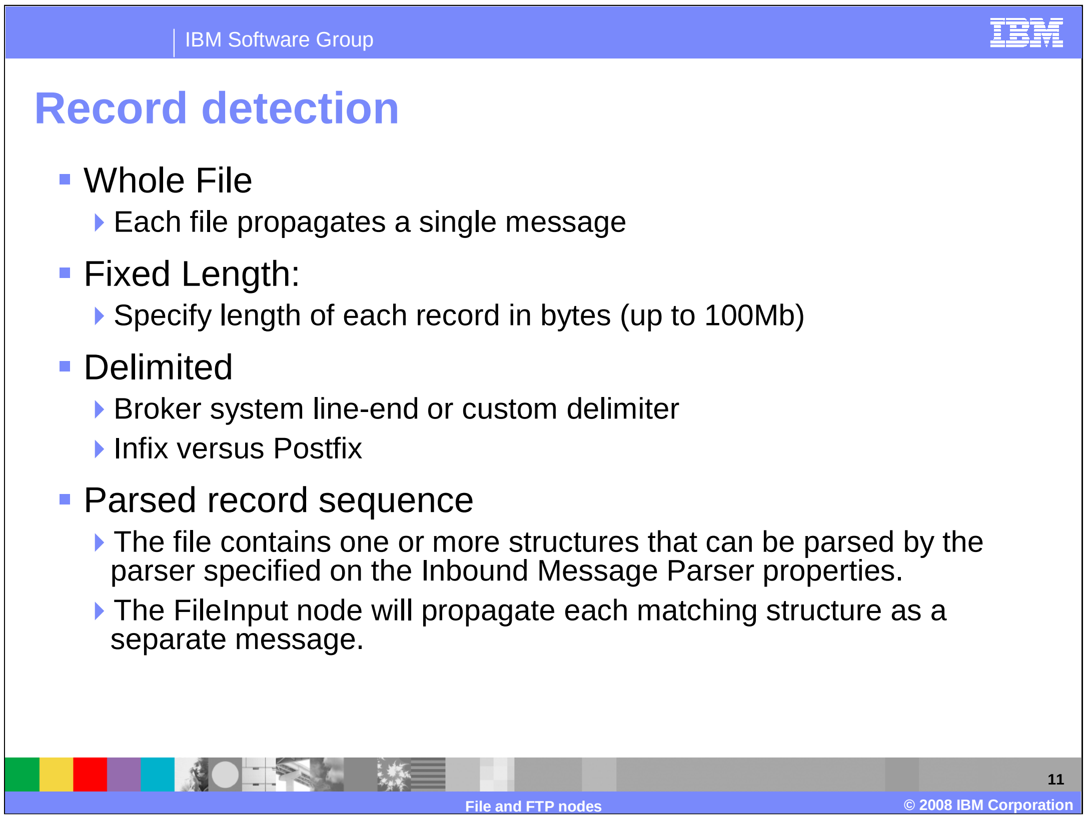
- The FileInput node can split each file up into separate records, which causes the message flow to be invoked multiple times
- Handling options (on the Records and Elements tab):
  - Whole file
  - Fixed Length
  - Delimited
  - Parsed Record Sequence
- Only requires one record to be in memory at any one time
  - Allows very large files to be streamed efficiently
  - Streaming possible with MRM (CWF and TDS) and XMLNSC parsers only
- If connected, the 'End Of Data' terminal is triggered after all the messages in a file have been processed.
  - Empty BLOB message and a LocalEnvironment.File structure

The File Input node allows you to split up each file into multiple records that are propagated separately, and the next few slides discuss the various options available for record parsing.

One particular benefit of record detection is that, depending on which parser you're using, the File Input node does not need to store the whole file in memory at any one time. Only the records currently being processed are stored in memory. This streaming allows for very large files to be processed. Such files could be several gigabytes in size. The capability is supported by the MRM and XMLNSC parsers.

Record detection allows files to be split into individual parts using several techniques. Alternatively, the whole file can be processed in one go. When processing records individually, the maximum size of an individual record is 100MB, which is the same as the maximum size of an MQ message. Note however, that processing records of this size will need careful planning of the Broker environment and in particular the size of the Broker Java Virtual Machine.

Note that a file can only be processed by a single thread within the Broker. Therefore, individual records can only be processed sequentially, not in parallel.



Record detection

- Whole File
  - Each file propagates a single message
- Fixed Length:
  - Specify length of each record in bytes (up to 100Mb)
- Delimited
  - Broker system line-end or custom delimiter
  - Infix versus Postfix
- Parsed record sequence
  - The file contains one or more structures that can be parsed by the parser specified on the Inbound Message Parser properties.
  - The FileInput node will propagate each matching structure as a separate message.

This slide describes the five options available for record detection.

"Whole File" reads the entire file's contents into memory and parses it in its entirety according to the Inbound Message Parsing options. If this option is used, then you must be sure that the available memory in the Broker is sufficient to accommodate the whole file.

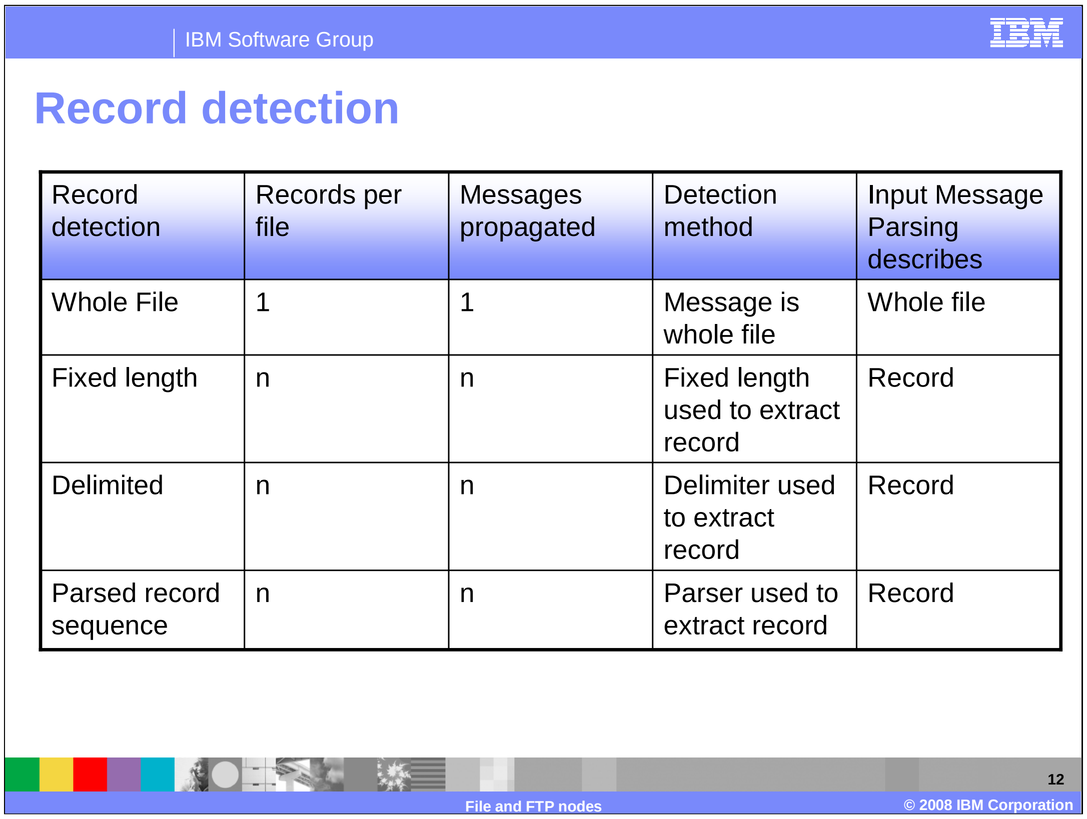
"Fixed Length" breaks the file up into a set number of bytes and propagates each chunk as a separate message. The last message in any file may be of a size smaller than the specified length. If you use this option, you specify the record size on the File Input Node

"Delimited" allows you to specify a delimiter that is used to break up the records in a file. In this case, you can specify a standard DOS or UNIX "end of line" delimiter, or you can specify you own custom delimiter. This is done by specifying the hexadecimal characters which indicate the end of line for the particular file.

If you specify *Delimiter type=Infix* and the last data in the file ends with a delimiter, the FileInput node will propagate an empty record to indicate the delimiter's presence. However, if you specify *Delimiter type=Postfix* the FileInput node will not propagate an empty record, whether the last data in the file ends with a delimiter or not.

There is a more advanced option for record detection.

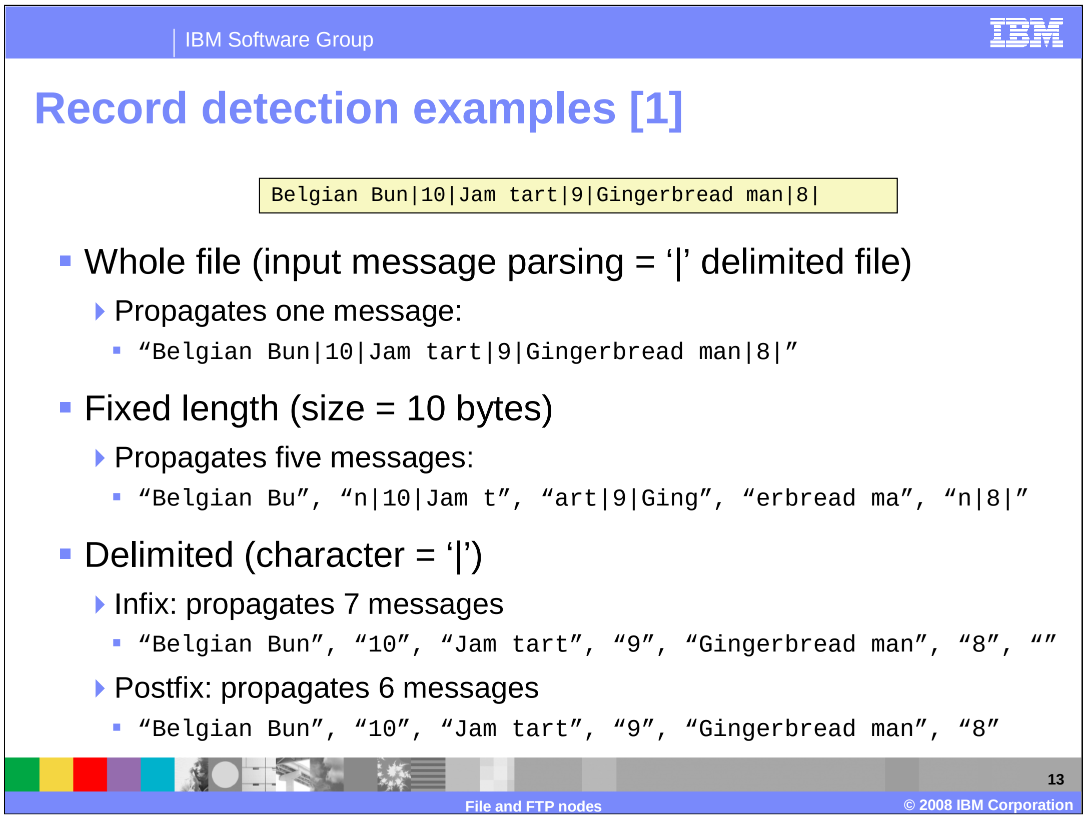
"Parsed Record Sequence" breaks the file into messages, each of which conforms to a single structure as defined on the Inbound Message Parsing options. For example, this could be used to process a file which contains multiple instances of a piece of XML data.



Record detection

Record detection	Records per file	Messages propagated	Detection method	Input Message Parsing describes
Whole File	1	1	Message is whole file	Whole file
Fixed length	n	n	Fixed length used to extract record	Record
Delimited	n	n	Delimiter used to extract record	Record
Parsed record sequence	n	n	Parser used to extract record	Record

This slide summaries the differences between the four options in tabular form.



IBM Software Group

Record detection examples [1]

```
Belgian Bun|10|Jam tart|9|Gingerbread man|8|
```

- Whole file (input message parsing = '|' delimited 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 7 messages
    - `"Belgian Bun", "10", "Jam tart", "9", "Gingerbread man", "8", ""`
  - Postfix: propagates 6 messages
    - `"Belgian Bun", "10", "Jam tart", "9", "Gingerbread man", "8"`

13

File and FTP nodes

© 2008 IBM Corporation

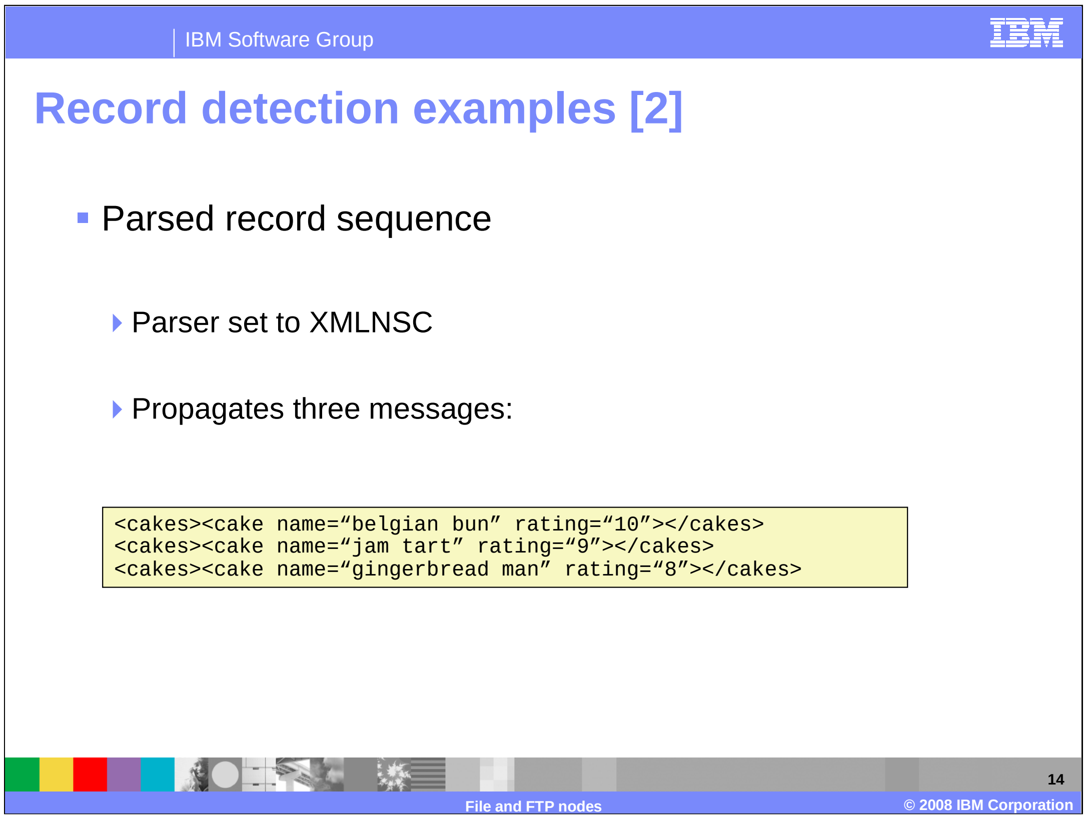
Using the contents of a simple file as an example, this slide describes how the different options affect what is propagated.

This example file is shown in the yellow box at the top of the slide. The file can be processed in different ways.

The first way is to process the file as "Whole File". In this case, the entire file is passed to the message flow as a single piece of data in the message tree.

The second way is to break the file into sections of 10 bytes each. In this case, the File Input node will propagate 5 messages to the flow. The first 4 messages will have 10 bytes each. The last message will have 4 bytes.

The third way is to treat the file with a Delimited Parser, using the vertical bar as a delimiter. In this case, if the "Delimiter Type" is set to "In-Fix", 7 messages will be propagated, where the last record is an empty record. If the "Delimiter Type" is set to "Post-Fix", 6 messages will be propagated.



Record detection examples [2]

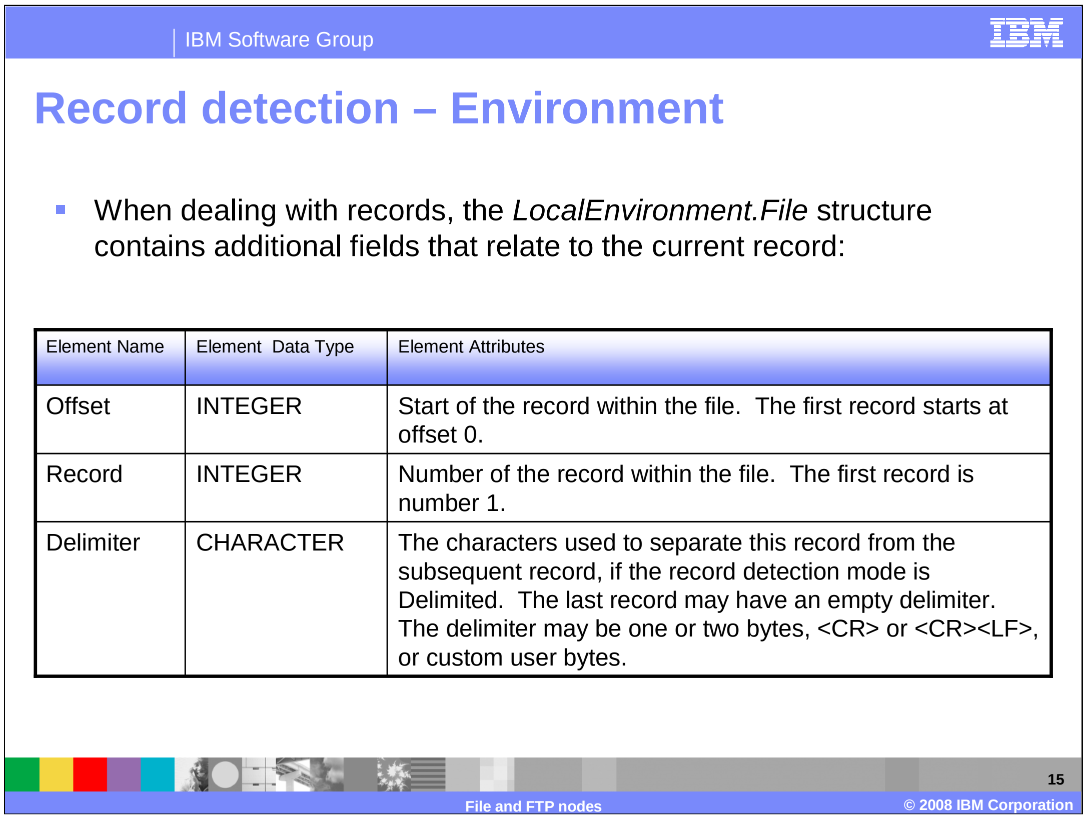
- Parsed record sequence
  - Parser set to XMLNSC
  - Propagates three messages:

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

These examples show how the two Parser-based options propagate messages.

For the *“Parsed Record Sequence”* example, note that the input file contents is not a single valid XML document, but rather three distinct XML documents which are propagated separately. On its own, the contents of the file would not pass a validation test of the XML, but when split into multiple instances of XML, then each single instance would be valid.

In this case, if the parser is set to XMLNSC, then three messages will be propagated to the message flow.



Record detection – Environment

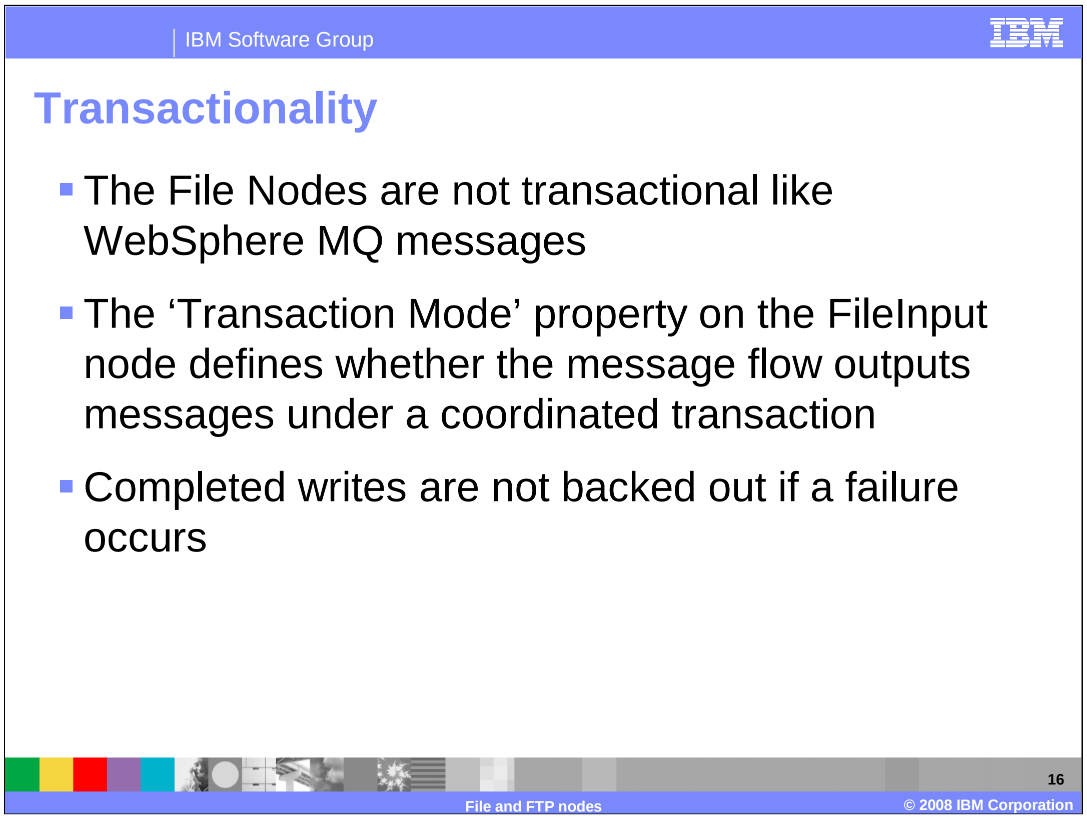
- When dealing with records, the *LocalEnvironment.File* structure contains additional fields that relate to the current record:

Element Name	Element Data Type	Element Attributes
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0.
Record	INTEGER	Number of the record within the file. The first record is number 1.
Delimiter	CHARACTER	The characters used to separate this record from the subsequent record, if the record detection mode is Delimited. The last record may have an empty delimiter. The delimiter may be one or two bytes, <CR> or <CR><LF>, or custom user bytes.

When parsing multiple records in a file, extra properties are stored in the LocalEnvironment, in the File part of that tree.

When using record detection = parsed embedded records, a LocalEnvironment.MessageContext element is also created. In this scenario, the FileInput node builds a tree in LocalEnvironment.MessageContext so that you can locate data from preceding headers, and check data in trailing footers, such as checksums and totals. The message context contains all the elements resulting from parsing the file up to and including the current record, except that preceding instances of the Propagated record elements are deleted, and the current element is empty. On the last element in the file, the message context is completed to the end of file.

You can consider the FileInput node as progressively parsing the whole file, building a tree under LocalEnvironment.MessageContext. Each time it reaches the Propagated record element, it moves its contents to the last child of Root, to create a message body, and propagates the result to its Out terminal. When the flow completes, the node deletes that message body, and the element (now empty) from the message context, and continues to parse for the next propagated message element. On the last occurrence, the parser completes the message context tree to the end of file.



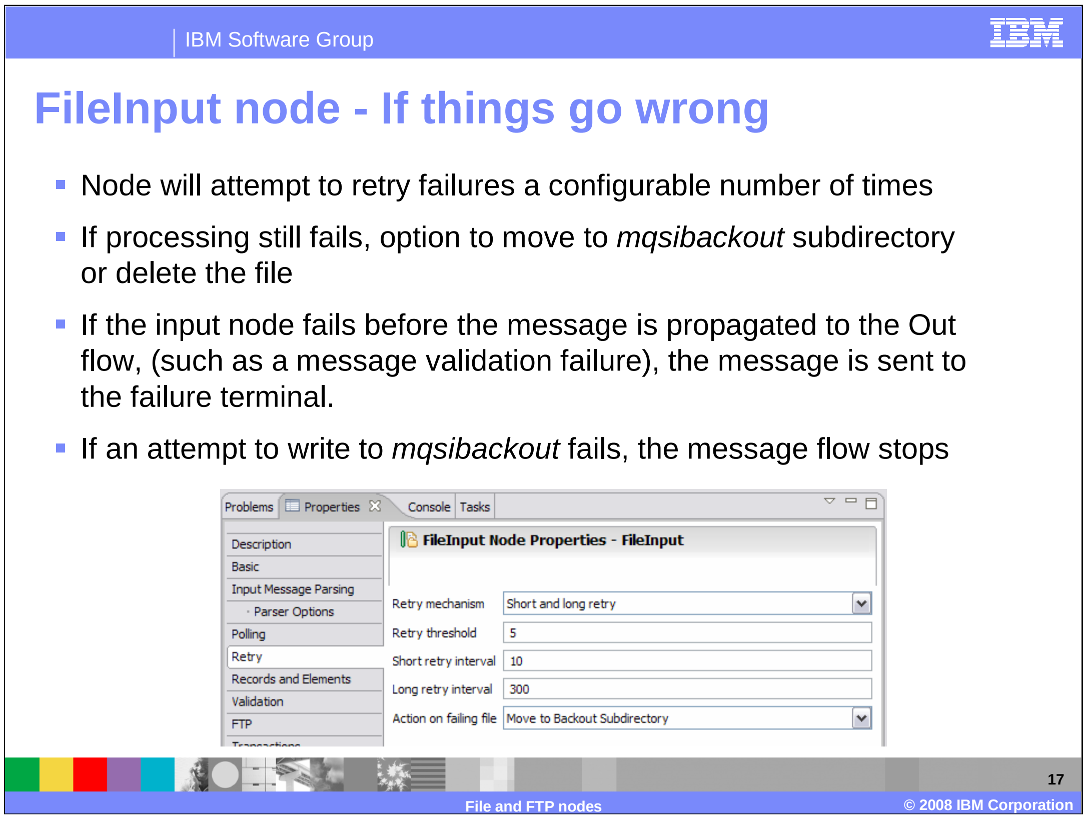
Transactionality

- The File Nodes are not transactional like WebSphere MQ messages
- The ‘Transaction Mode’ property on the FileInput node defines whether the message flow outputs messages under a coordinated transaction
- Completed writes are not backed out if a failure occurs

When you include a File Input node in a message flow, the value that you set for Transaction Mode defines whether messages are sent under sync point. The File Input node is not transactional, and does **not itself** process files under sync point.

If you set “*Transaction Mode”* to Yes, the message is received under sync point (that is, within a WebSphere MQ unit of work). Any messages subsequently sent by an output node in the same instance of the message flow are put under sync point, unless the output node has explicitly overridden this. If the File Input node backs-out the transaction, all changes made under sync point are backed out.

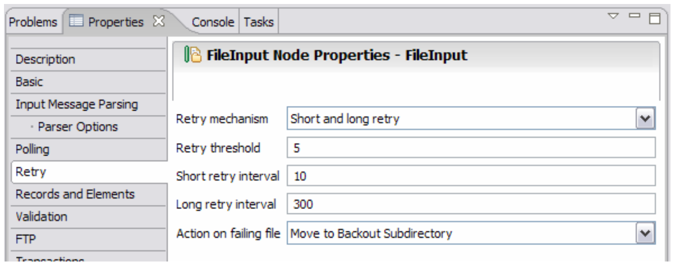
If you set “*Transaction Mode”* to No, the file is not received under sync point. Any messages subsequently sent by an output node in the flow are not put under sync point, unless an individual output node has specified that the message must be put under sync point. Changes not made under sync point will not be backed-out if the File Input node backs out the transaction.



IBM Software Group

FileInput node - If things go wrong

- Node will attempt to retry failures a configurable number of times
- If processing still fails, option to move to *mqsibackout* subdirectory or delete the file
- If the input node fails before the message is propagated to the Out flow, (such as a message validation failure), the message is sent to the failure terminal.
- If an attempt to write to *mqsibackout* fails, the message flow stops



FileInput Node Properties - FileInput

Retry mechanism	Short and long retry
Retry threshold	5
Short retry interval	10
Long retry interval	300
Action on failing file	Move to Backout Subdirectory

17

File and FTP nodes

© 2008 IBM Corporation

Each FileInput node has a configurable retry mechanism. There are three options for *"Retry Mechanism"* .

The first option is *Failure*, which is the default action. If the file could not be read, then it should be processed as a failure.

The second option is *Short Retry*. In this case, the Broker retries several times before processing as a failure. The number of times to retry is given by "*Retry threshold*" and the "*Short retry interval"* specifies the length of time to wait between failures, in seconds.

The third option is "*Short retry then Long retry*". In this case, once short retries have been exhausted, the node will continue to retry indefinitely, waiting for the Long retry interval period between retries.

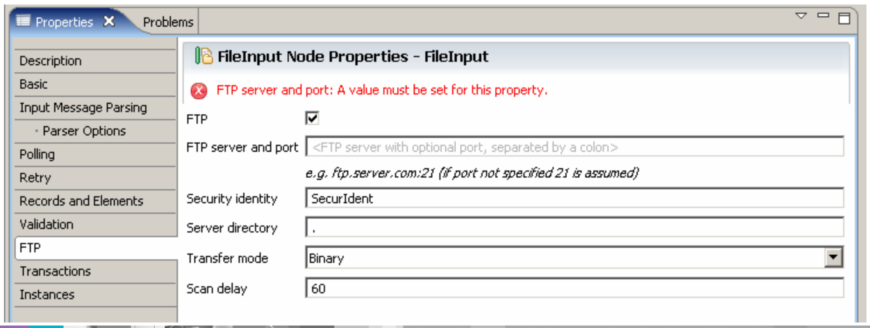
Failure processing is determined by the "Action on failing file" option. This provides three options.

The first option is "*Move to Backout Subdirectory".* In this case, the file is written to a back out subdirectory, which is named "mqsi-backout". This directory is created as a subdirectory of the input directory. This is created automatically if it doesn't already exist. If a file of that name already exists in the back out directory, the message flow fails; the failure will be reported in the normal manner, for example using the Event Log in a Windows environment.

The second option is "*Delete*". In this case, the input file is deleted and the next file is processed.

FileInput node – using FTP

- 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 ftpUser –p ftpPassword`



FileInput Node Properties - FileInput

FTP server and port: A value must be set for this property.

FTP: ☑

FTP server and port: <FTP server with optional port, separated by a colon>

e.g. ftp.server.com:21 (if port not specified 21 is assumed)

Security identity: SecurIdent

Server directory: .

Transfer mode: Binary

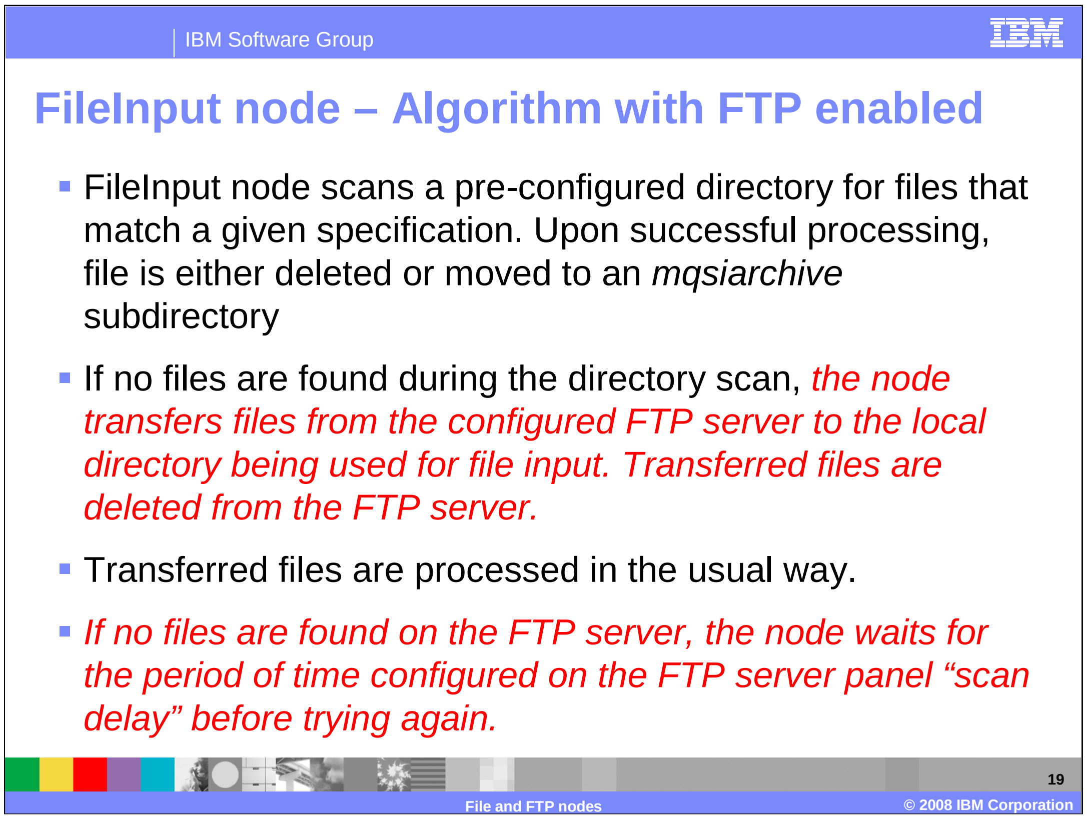
Scan delay: 60

It is possible to use the File Input to read files from an FTP server. FTP support is enabled through a single Boolean flag on the FTP tab of the File Input node. When using the FTP function, the Broker operates in a similar way to normal file processing. The FTP server directory is scanned for new files, and any files that are detected are moved into the local file directory specified on the File Input Node. These files are then processed by the Broker as a local file.

The security authorization for the FTP server can be specified using the *"Security Identity"* value of the FTP tab. The value specified in here should match the parameter that is specified on the "mqsi-set-db-parms" command.

When using the "mqsi-set-db-parms" command, note the format of the "ftp" parameter. This must be followed by a double colon, and then the name of the *Security Identity* property, which is used on the FTP tab of the File Node.

IPv6 format addresses can be used as the FTP server address, although the address must be enclosed in square brackets.


IBM Software Group

FileInput node – Algorithm with FTP enabled

- FileInput node scans a pre-configured directory for files that match a given specification. Upon successful processing, file is either deleted or moved to an *mqsiarchive* subdirectory
- If no files are found during the directory scan, *the node transfers files from the configured FTP server to the local directory being used for file input. Transferred files are deleted from the FTP server.*
- Transferred files are processed in the usual way.
- *If no files are found on the FTP server, the node waits for the period of time configured on the FTP server panel "scan delay" before trying again.*

19

File and FTP nodes

© 2008 IBM Corporation

Typically, FTP connections require a longer poll interval than local directory scans. The use of separate interval properties for FTP and local files makes it possible to enable and disable FTP function through a single "Enable FTP" property, rather than reconfiguring the node's interval property each time the capability is enabled or disabled.

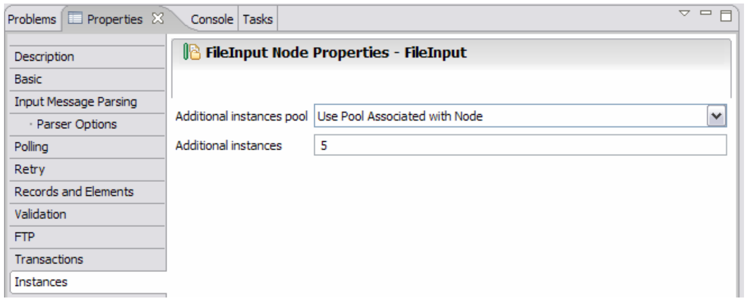
The changes to the basic algorithm for the FTP scenario are shown in red on this slide. So, the FTP Node will scan the local file directory as before. If no files are found, it will use the FTP function to connect to the specified FTP server, and retrieve all files in the target FTP directory.

Note that transferred files are deleted from the FTP server once they have been transferred to the local directory. This is to avoid re-processing of files from the FTP server. However, if you need to keep a copy of the transferred file, this can be copied into the local archive directory, using normal File processing.

If no files are found, the FTP "scan delay" is activated, before the directory is rescanned. This FTP Scan Delay overrides the polling interval that can be specified for local files. This is a separate property because it is likely that you would want to wait for a longer interval when using FTP, compared to a delay for a scan of a local file system.

Message Broker Version 6.1 supports both Active and Passive FTP Transfers.

Additional instances handling



FileInput Node Properties - FileInput

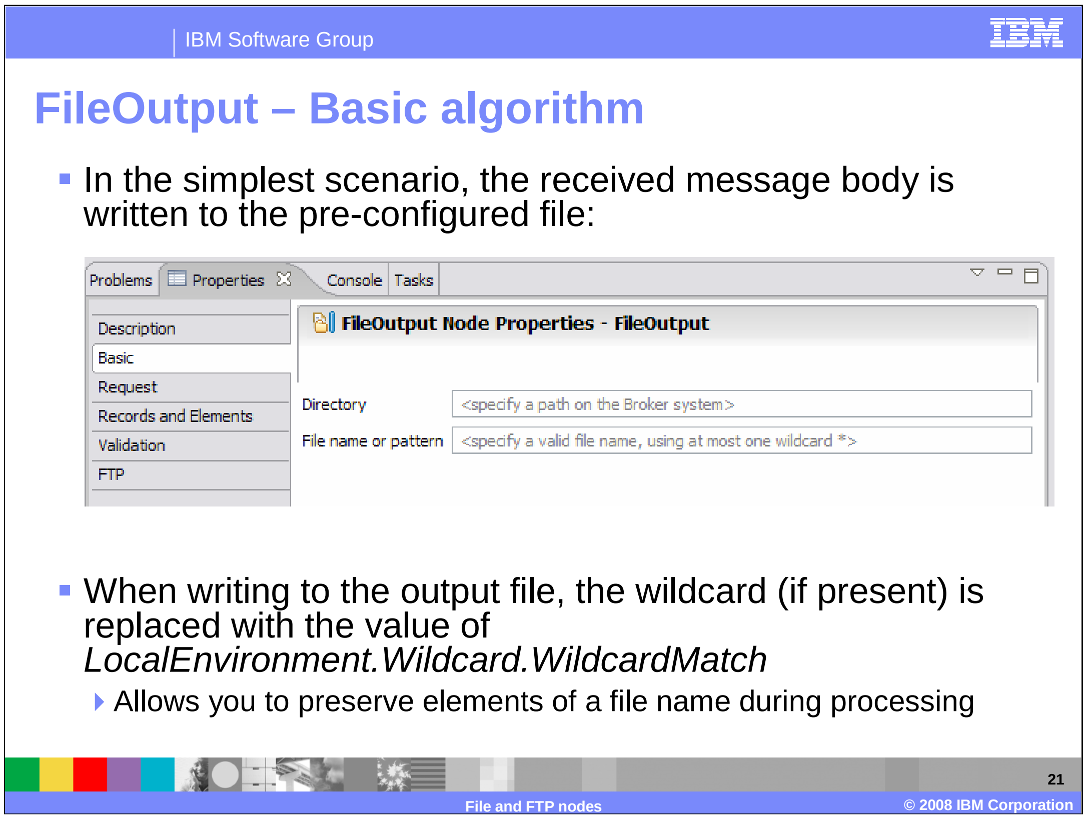
Additional instances pool: Use Pool Associated with Node

Additional instances: 5

- Each file is processed by at most one instance and node
- Additional instances and nodes can concurrently process other files in the same directory.
- Option to allocate threads from a node-specific pool rather than the flow pool
- Additional instances tuning is particularly important when FTP is involved

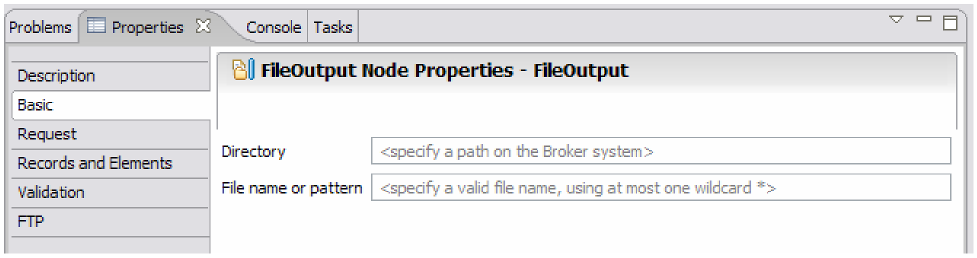
Several input nodes now have an Instances tab which allows you to enable the use of a dedicated thread pool for the node. This is useful in message flows that have multiple input nodes, as the flow’s pool of threads can lead to starvation of one or more input nodes.

In particular, FTP transfers may take a significant amount of time to complete, during which time the thread is busy. For this reason it is important to tune the additional instances parameter in order to maximize throughput rates.



FileOutput – Basic algorithm

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



FileOutput Node Properties - FileOutput

Directory: <specify a path on the Broker system>

File name or pattern: <specify a valid file name, using at most one wildcard *>

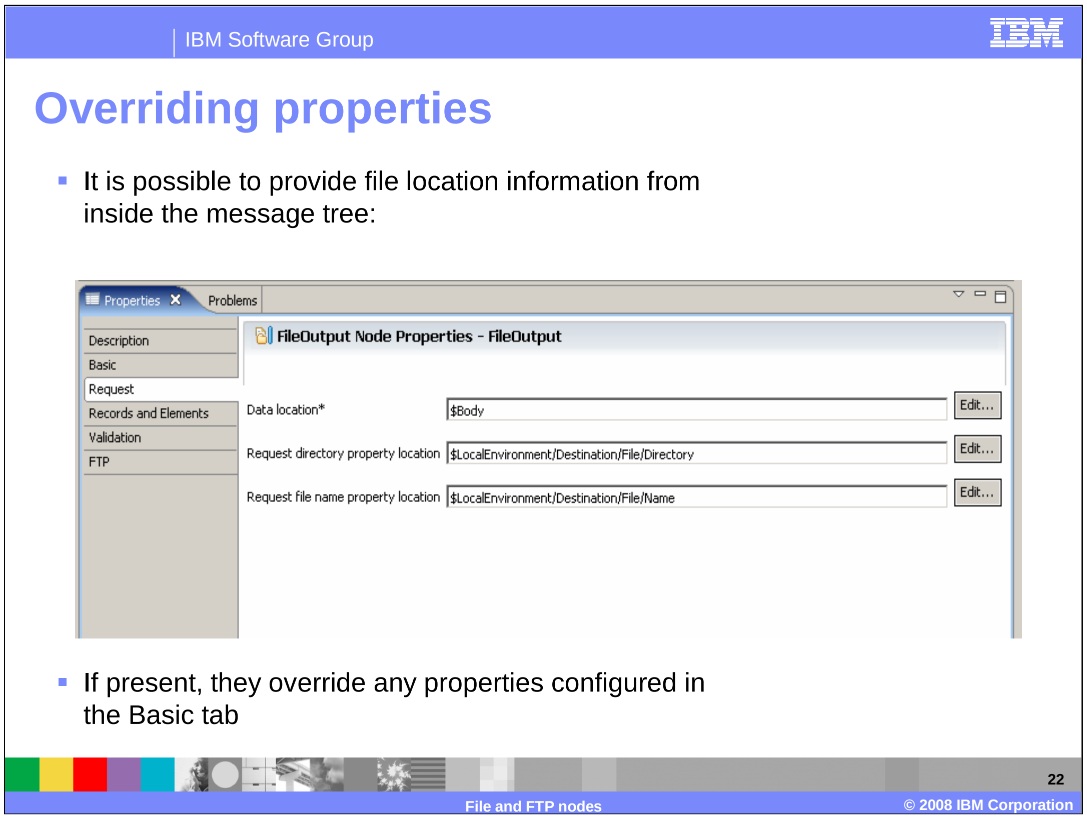
- When writing to the output file, the wildcard (if present) is replaced with the value of *LocalEnvironment.Wildcard.WildcardMatch*
  - Allows you to preserve elements of a file name during processing

The following slides discuss the File Output node.

The File Output Node is similar to the File Input node, in that in its simplest form you can specify the directory to which files are placed and a file name expression that describes the file name.

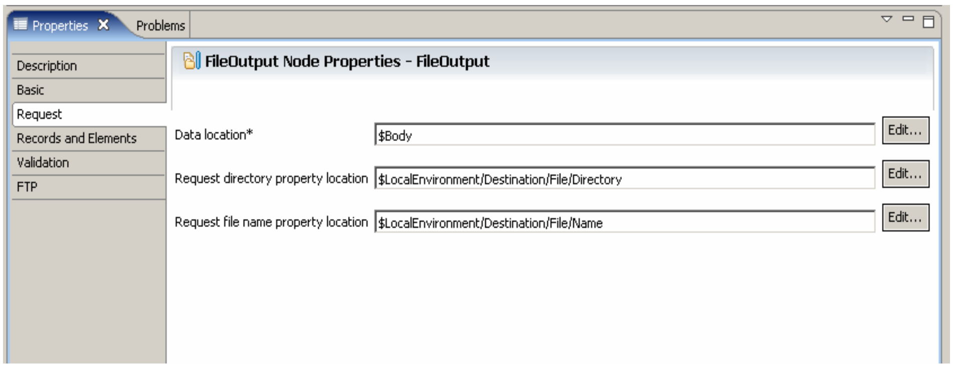
The file name expression can include a single wildcard character '*', which will be replaced with the value of the *LocalEnvironment.Wildcard.WildcardMatch* element in the tree.

For example, in a File Input to File Output scenario, the Wildcard-Match element is set up during the File Input node processing. This allows you to preserve the name of the input file on the File Output node. You can also use standard tree manipulation logic to set the value of the Wildcard-Match element to whatever you want, regardless of whether you used a File Input node or not. For example, this could be used to specify the name of the output file.



Overriding properties

- It is possible to provide file location information from inside the message tree:



FileOutput Node Properties - FileOutput

Data location* $Body

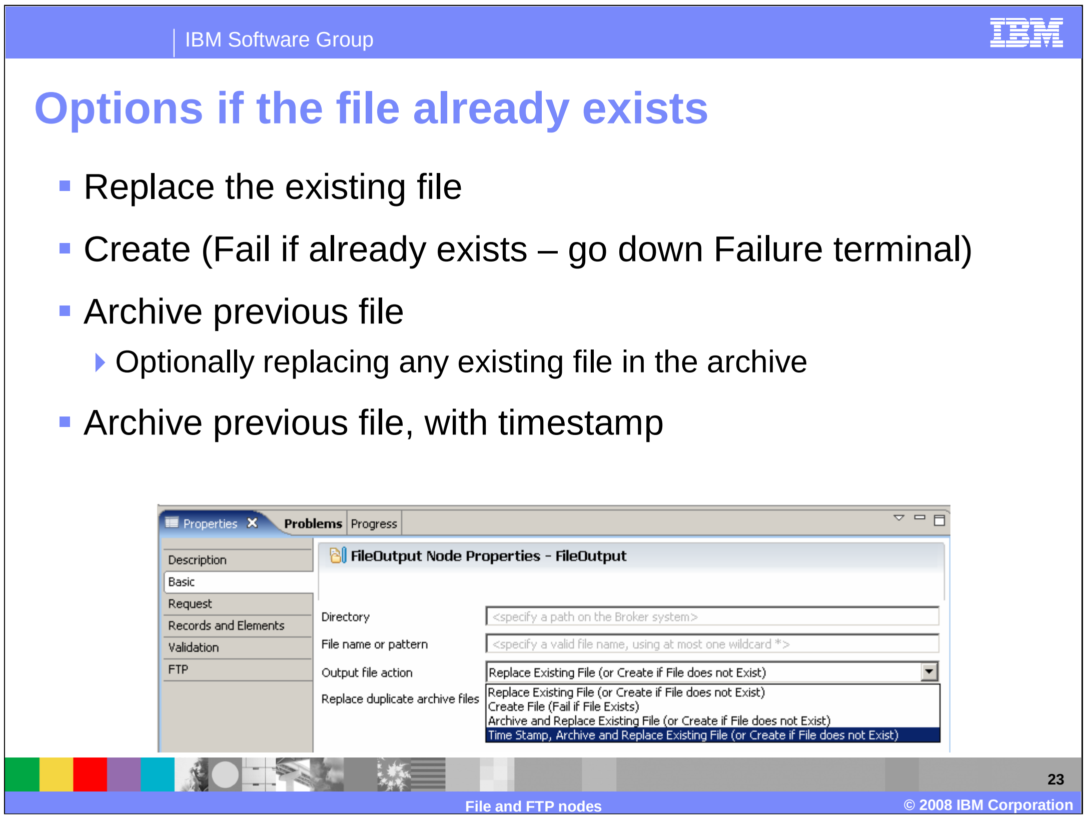
Request directory property location $LocalEnvironment/Destination/File/Directory

Request file name property location $LocalEnvironment/Destination/File/Name

- If present, they override any properties configured in the Basic tab

The tree element that is serialized to the output file is obtained from the value of the ‘Data location’ property on the Request tab. The default value is $Body.

It is also possible to override the target directory and file name information. By default, this information is obtained from the local environment, although this can be any place in the message tree. These parameters will override any hard-coded values that you may have specified on the “Basic” tab.



Options if the file already exists

- Replace the existing file
- Create (Fail if already exists – go down Failure terminal)
- Archive previous file
  - Optionally replacing any existing file in the archive
- Archive previous file, with timestamp

FileOutput Node Properties - FileOutput

Directory <specify a path on the Broker system>
File name or pattern <specify a valid file name, using at most one wildcard *>
Output file action Replace Existing File (or Create if File does not Exist)
Replace duplicate archive files

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)

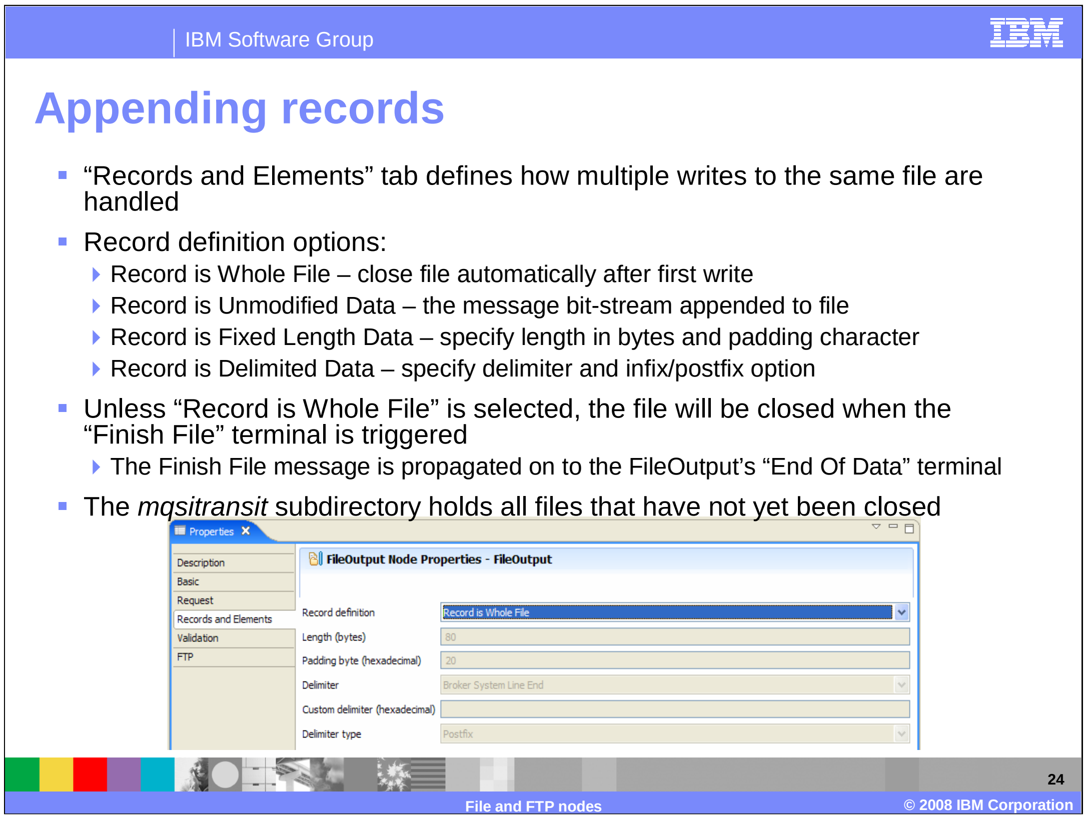
This slide describes the options that are available when writing to a file of a name that already exists in the output directory.

The first option is to Replace the existing file. This will over-write the existing file, with no further warning.

The second option is to Create the file normally, but fail if the file already exists. Control would be passed to the Failure output terminal in this instance.

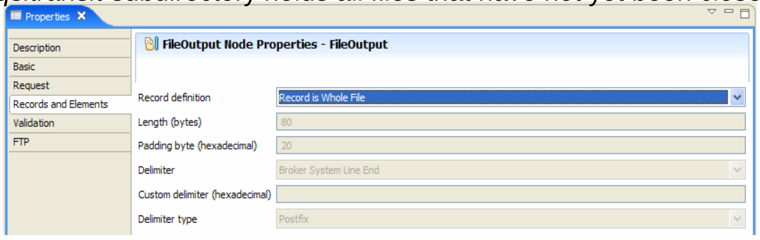
The third option is to Archive any existing file, before over-writing it. In this case, the original file would be copied to the archive directory.

The final option is the same as the third option, but you can also change the name of the original file by adding a Time Stamp to the file name. As with the Input Node options, this will avoid over-writing an existing file, since the time-stamp will be unique.



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

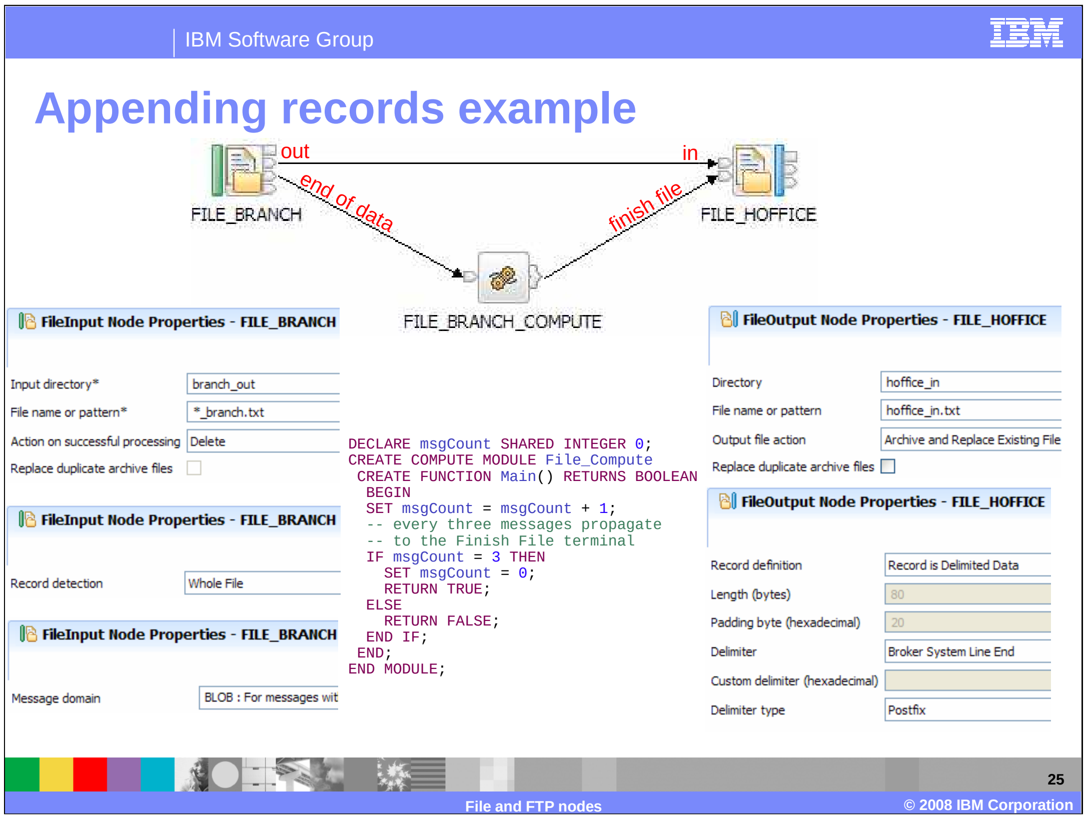


FileOutput Node Properties - FileOutput

Record definition	Record is Whole File
Length (bytes)	80
Padding byte (hexadecimal)	20
Delimiter	Broker System Line End
Custom delimiter (hexadecimal)	
Delimiter type	Postfix

If append record is enabled, a separate set of options (on the "Records and Elements" tab) define how the record is appended and this slide describes those options.

For the delimited record option, infix states that the last record in a file will be followed by a delimiter.



IBM Software Group

Appending records example

FileInput Node Properties - FILE_BRANCH

Input directory*	branch_out
File name or pattern*	*_branch.txt
Action on successful processing	Delete
Replace duplicate archive files	☐

FileInput Node Properties - FILE_BRANCH

Record detection	Whole File

FileInput Node Properties - FILE_BRANCH

Message domain	BLOB : For messages wit

```
DECLARE msgCount SHARED INTEGER 0;
CREATE COMPUTE MODULE File_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
  SET msgCount = msgCount + 1;
  -- every three messages propagate
  -- to the Finish File terminal
  IF msgCount = 3 THEN
    SET msgCount = 0;
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
 END;
END MODULE;
```

FileOutput Node Properties - FILE_HOFFICE

Directory	hoffice_in
File name or pattern	hoffice_in.txt
Output file action	Archive and Replace Existing File
Replace duplicate archive files	☐

FileOutput Node Properties - FILE_HOFFICE

Record definition	Record is Delimited Data
Length (bytes)	80
Padding byte (hexadecimal)	20
Delimiter	Broker System Line End
Custom delimiter (hexadecimal)	
Delimiter type	Postfix

25

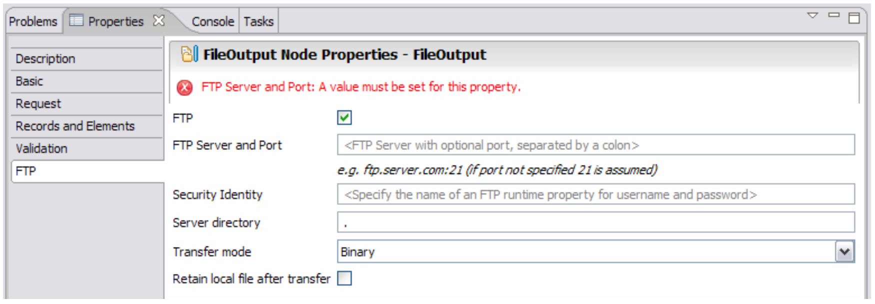
File and FTP nodes © 2008 IBM Corporation

This example shows how to append records to a file that is being processed by a File Input Node, and written using a File Out put Node.

The slide shows the properties of both the input and output nodes, together with the ESQL code that is contained in the intermediate Compute Node.

When the end of the input file is reached, the “End of Data” terminal is activated, and the message flow continues to the Compute Node. The Compute Node appends three records to the file, which are then propagated to the Output Node.

FileOutput - FTP support



Problems | Properties | Console | Tasks

Description
Basic
Request
Records and Elements
Validation
FTP

FileOutput Node Properties - FileOutput

FTP Server and Port: A value must be set for this property.

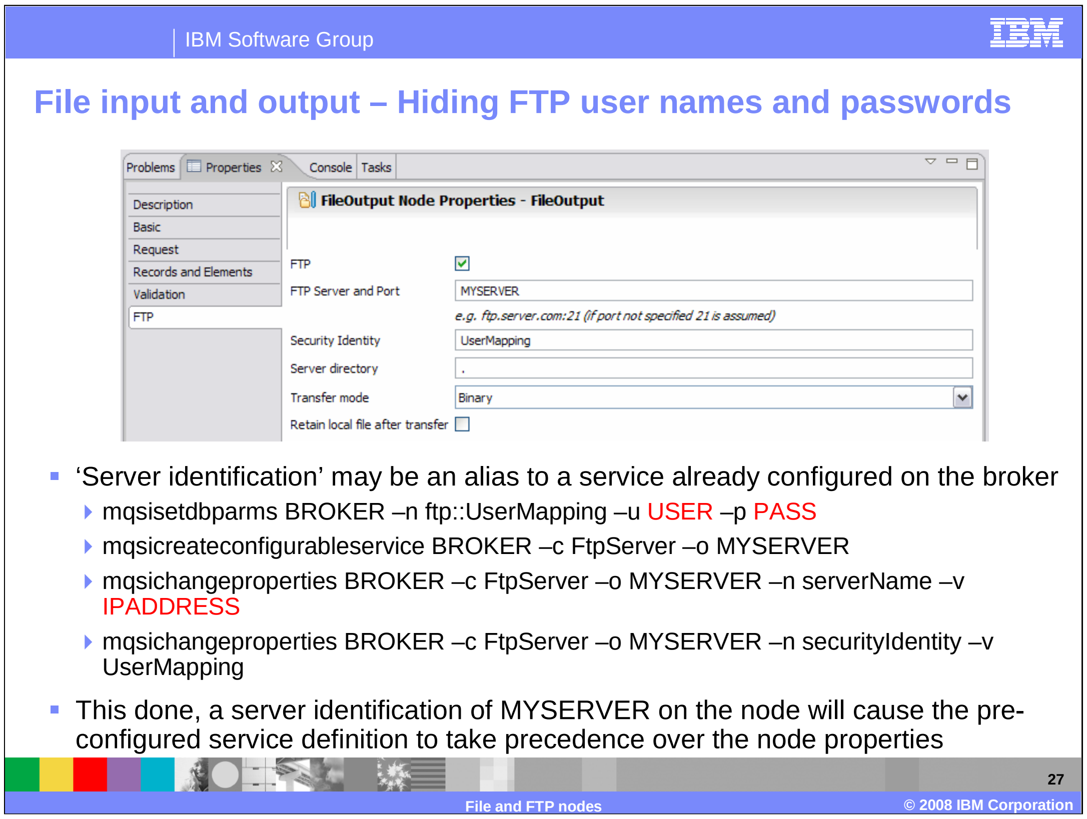
FTP ✓
FTP Server and Port: <FTP Server with optional port, separated by a colon>
e.g. ftp.server.com:21 (if port not specified 21 is assumed)
Security Identity: <Specify the name of an FTP runtime property for username and password>
Server directory: .
Transfer mode: Binary
Retain local file after transfer ☐

- 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

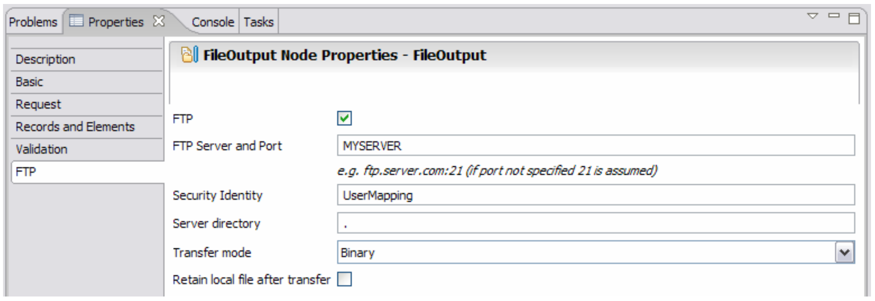
As with the File Input node, FTP support is configurable on a single ‘FTP’ tab and enabled through the use of a single check box on the FTP Properties tab.

FTP transfer only happens after a file is closed on the local file system. After the file is transferred to the FTP server, it can be retained locally, or deleted from the local file system.

The FTP process happens synchronously, and can take an extended time, depending on the available network bandwidth, and the size of the file to be transferred. As with the FTP Input function, you should make use of the “additional instances” property, to make sure that thread starvation does not occur.



File input and output – Hiding FTP user names and passwords



FileOutput Node Properties - FileOutput

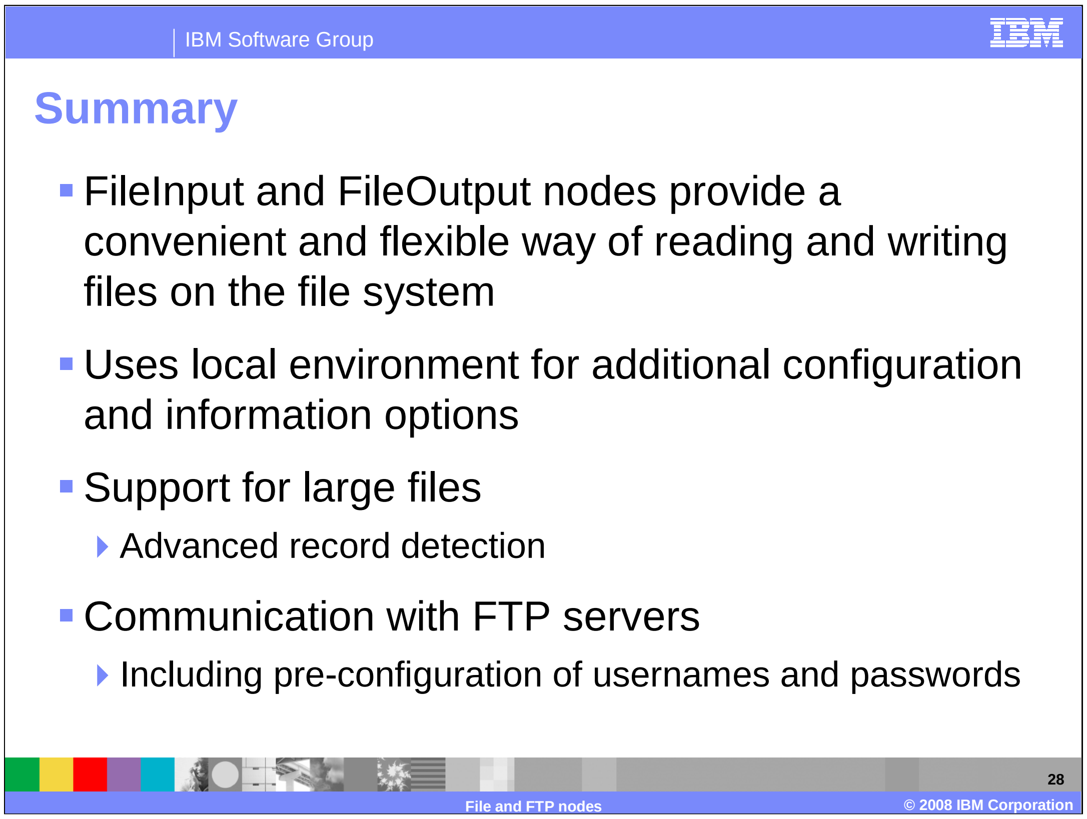
FTP ✓
FTP Server and Port: MYSERVER
e.g. ftp.server.com:21 (if port not specified 21 is assumed)
Security Identity: UserMapping
Server directory: .
Transfer mode: Binary
Retain local file after transfer ☐

- 'Server identification' may be an alias to a service already configured on the broker
  - mqsisetdbparms BROKER –n ftp::UserMapping –u USER –p PASS
  - mqsicreateconfigurableservice BROKER –c FtpServer –o MYSERVER
  - mqsichangeproperties BROKER –c FtpServer –o MYSERVER –n serverName –v IPADDRESS
  - mqsichangeproperties BROKER –c FtpServer –o MYSERVER –n securityIdentity –v UserMapping
- This done, a server identification of MYSERVER on the node will cause the pre-configured service definition to take precedence over the node properties

Specification of the FTP Security parameters is configured in the same way as on the Input Node. The FTP user ID and password are not stored within the message flow, but in the Broker runtime, using the "mqsi-set-db-parms" command.

It is also possible to specify the target FTP server using the "mqsi-create-configurable-service" command. One of the parameters on this command refers to the security identification created earlier.

Using a combination of these commands enables an FTP service to be fully configured outside the message flow. This can then be referenced in the File Output Node properties, as shown on this slide.



IBM Software Group

Summary

- FileInput and FileOutput nodes provide a convenient and flexible way of reading and writing files on the file system
- Uses local environment for additional configuration and information options
- Support for large files
  - Advanced record detection
- Communication with FTP servers
  - Including pre-configuration of usernames and passwords

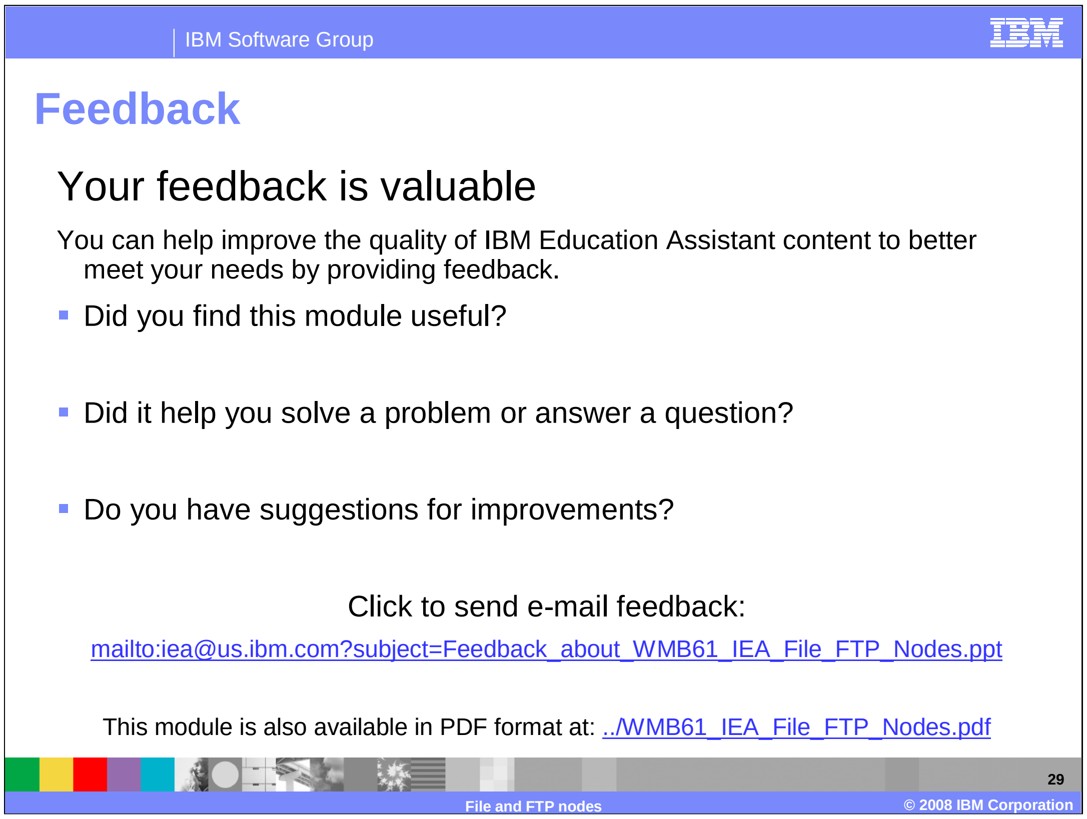
28

File and FTP nodes

© 2008 IBM Corporation

This slide summarizes the capability provided by the file nodes in Message Broker version 6.1.

The File Input and File Output Nodes provide capability to process both local and remote files. These files can be very large, and files can be processed using normal Message Broker parsing capability.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WMB61_IEA_File_FTP_Nodes.ppt

This module is also available in PDF format at: ../WMB61_IEA_File_FTP_Nodes.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers