Generic Message Handler (GeMHa)

Table of Contents

Description....3
  Application Configuration File....6
    Input....7
    Output....8
    Processing....8
Description of Processors....13
  DoNothing....13
  ForDB....14
    INSERTs....14
    UPDATEs....15
    DELETEs....15
    SELECTs....16
    Prepared Statements....18
    ForDb Configuration File....19
  ForSocket....22
  ForFile....25
General Features....28
  Logging....28
  Audit Trail....28
  XML Validation....28
  Resource Management....28
Examples....29
  Example 1 – Database Query....29
    Application Configuration File (Applic_FtoDBtoF_001.xml)....29
    ForDb Configuration File (ProcessMessageForDbSettings.xml)....30
    Input Data (select_001.xml)....31
    Execution....32
    Response Data (response_1002_201310031133250.xml)....32
  Example 2 – Database Inserts....33
    Application Configuration File (Applic_CtoDBtoF.xml)....33
    ForDb Configuration File (ProcessMessageForDbSettings.xml)....34
    Input Data (input_CSV_001.txt, input_CSV_002.txt)....34
    Execution....34
    Response Data....35

Description

The Message Handler is an application facilitating communication between applications, including across servers, usually in the form of XML messages. It also allows presentation of simplified database actions (SELECT, INSERT, UPDATE, DELETE) to a specified database and return of a response, all via XML messages.

It takes inputs from one of ...

- a message **Queue**
- a **socket** connection
- a **CSV** file of input records
- a set of message **files**

...hands the input message to a specified ***MessageProcessor*** object, which ensures the message is delivered/processed, creates a response and places the response...

- on a message **Queue**
- on an **HTTP** wire
- in a **file**

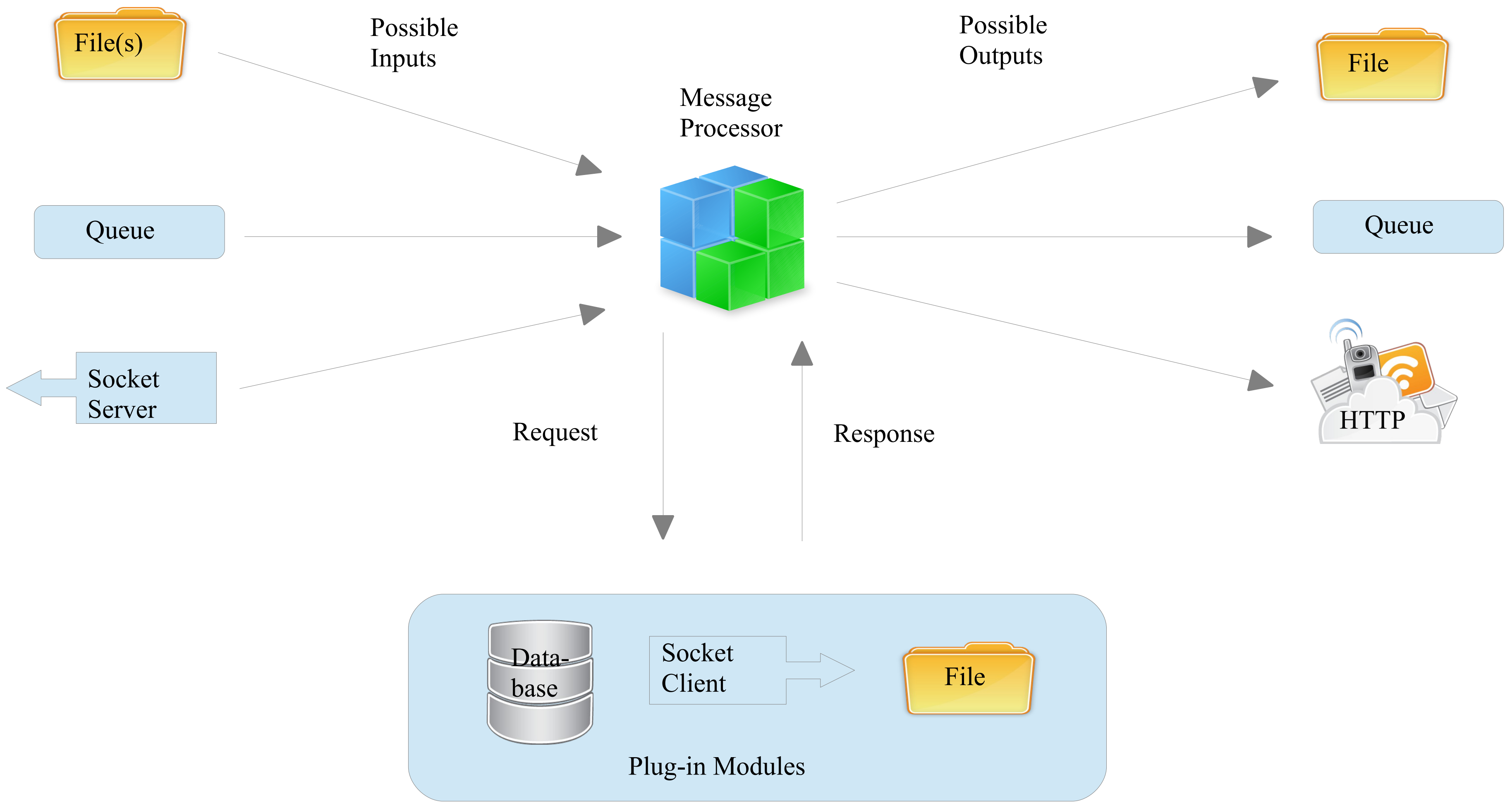
The Message Handler is designed to allow you plug in one of three ***MessageProcessors.*** These are java classes implementing a defined *IProcessMesssage* interface. This interface has defined methods for performing start-up, message-processing, and close-down operations, which are called by the Message Handler. The currently available plug-ins perform the following :

- Perform simple Inserts, Updates, Deletes or Queries to a **database**
- Send the message over a **socket** connection
- Send the messages as **CSV** records to an output file

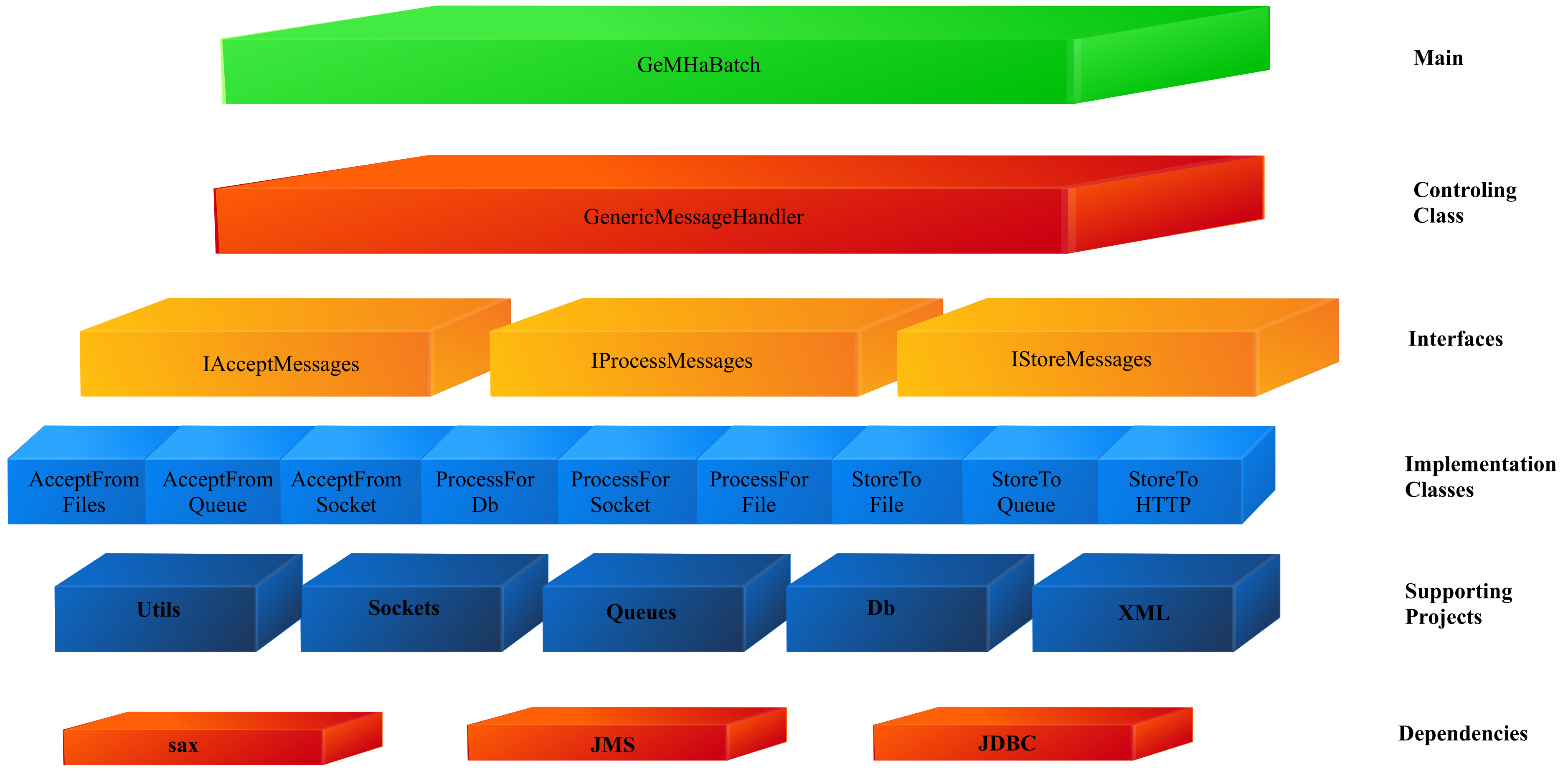
By mixing and matching the various modules, messages can be transferred from a Queue, Socket or File(s), to a Database, Queue, Socket, File(s) or HTTP.

The following two pages show a diagrammatic representation of the possible Message Handler configurations and the design of the different modules of the application.

GeMHa Processing Configurations



GeMHa Modules



Application Configuration File

Which modules are instantiated in any given run of the application is defined in the *Application Configuration* file, which is passed as the only argument when executing on the command line.

File `WebContent/unprotected/gemha_config.html` runs a Wizard to help you generate an *Application Configuration* file.

The following table describes the structure of the *Application Configuration* file...

Tag Name	Parent	Occurrence	Type	Domain / Format	Default
Applic		1	Aggregate		
Params	Applic	0..1	Aggregate		
ExternalShell	Applic/Params	0..1	string		"sh"
Input	Applic	1	Aggregate		
InputSource	Applic/Input	1	Aggregate		
InputLimit	Applic/Input/InputSource	0..1	integer	1..n	0
DataFormat	Applic/Input/InputSource	1	DataFormatType		
DataContractName	Applic/Input/InputSource	0..1	string		
InputQueue	Applic/Input/InputSource	0..1	Aggregate		
QueueManagerName	Applic/Input/InputSource/InputQueue	1	string		
QueueName	Applic/Input/InputSource/InputQueue	1	string		
MilliSecondsBeforeQuiet	Applic/Input/InputSource/InputQueue	0..1	integer	1..n	
InputSocket	Applic/Input/InputSource	0..1	Aggregate		
PortNumber	Applic/Input/InputSource/InputSocket	1	integer	1..n	
InputFile	Applic/Input/InputSource	0..1	Aggregate		
FileNameFilter	Applic/Input/InputSource/InputFile	1	string		
FileDir	Applic/Input/InputSource/InputFile	0..1	string		current directory
SortFilteredFileNames	Applic/Input/InputSource/InputFile	0..1	boolean	true/false	TRUE
CSVParams	Applic/Input/InputSource/InputFile	0..1	Aggregate		
FieldSeparator	Applic/Input/InputSource/InputFile/CSVParams	0..1	string		tab character
MaxRecsPerMessage	Applic/Input/InputSource/InputFile/CSVParams	0..1	integer	1..n	1
NumRecordsToSkip	Applic/Input/InputSource/InputFile/CSVParams	0..1	integer	0..n	0
ColumnOrder	Applic/Input/InputSource/InputFile/CSVParams	1	Aggregate		
Column	Applic/Input/InputSource/InputFile/CSVParams/ColumnOrder	1..n	string		
XMLFormat	Applic/Input/InputSource/InputFile/CSVParams	1	DbActionType		
InsertParams	Applic/Input/InputSource/InputFile/CSVParams	0..1	Aggregate		
Action_On_Error	Applic/Input/InputSource/InputFile/CSVParams/InsertParams	0..1	Action_On_Error Type		
Prepared_Statement_Name	Applic/Input/InputSource/InputFile/CSVParams/InsertParams	0..1	string		
Immediate_Commit	Applic/Input/InputSource/InputFile/CSVParams/InsertParams	0..1	Immediate_CommitType		
InputValidation	Applic/Input/InputSource	0..1			
Output	Applic	0..1	Aggregate		
OutputQueue	Applic/Output	0..1	Aggregate		
QueueManagerName	Applic/Output/OutputQueue	1	string		
QueueName	Applic/Output/OutputQueue	1	string		
ReplyToQueueManagerName	Applic/Output/OutputQueue	0..1	string		
ReplyToQueueName	Applic/Output/OutputQueue	0..1	string		
OutputFile	Applic/Output	0..1	Aggregate		
FileNameTemplate	Applic/Output/OutputFile	1	string		
OutputHTTP	Applic/Output	0..1			
ServerUrl	Applic/Output/OutputHTTP	1	string		
EndPointName	Applic/Output/OutputHTTP	1	string		
HTTPWithBackoff	Applic/Output/OutputHTTP	0..1	boolean		TRUE

Tag Name	Parent	Occurrence	Type	Domain / Format	Default
Processing	Applic	1	Aggregate		
MessageProcessingClassName	Applic/Processing	1	string		
MessageProcessingSettingsFileName	Applic/Processing	0..1	string		
MinResponsesExpected	Applic/Processing	0..1	integer	0..n	0
MindElements	Applic/Processing	0..1	Aggregate		
ElementName	Applic/Processing/MindElements	1..n	string		
SendElements	Applic/Processing	1	Aggregate		
ElementName	Applic/Processing/SendElements	1..n	string		
TargetMainDocElementName	Applic/Processing	0..1	string		"MESSAGE"
ResponseMainDocElementName	Applic/Processing	0..1	string		"MESSAGE"
ResponseLiterals	Applic/Processing	1	Aggregate		
ResponseLiteral	Applic/Processing/ResponseLiterals	1..n	string		
Auditing	Applic/Processing	0..1	Aggregate		
AuditKeys	Applic/Processing/Auditing	0..1	Aggregate		
KeyName	Applic/Processing/Auditing/AuditKeys	1..n	string		
AuditKeysSeparator	Applic/Processing/Auditing	0..1	string		
ErrorFiles	Applic/Processing/Auditing	0..1	Aggregate		
ErrorFilesDir	Applic/Processing/Auditing/ErrorFiles	0..1	string		current directory
ErrorFileNameTemplate	Applic/Processing/Auditing/ErrorFiles	0..1	string		ErrorMessage_*_?.txt
Logging	Applic/Processing	1	Aggregate		
Level	Applic/Processing/Logging	0..1	Aggregate		
EnvironmentLoggingLevel	Applic/Processing/Logging/Level	0..1	SystemEnvironmentType		
GeneralLoggingLevel	Applic/Processing/Logging/Level	0..1	LoggingLevelType		
LogFileDir	Applic/Processing/Logging	1	string		
LogFileNameTemplate	Applic/Processing/Logging	1	string		"GenericMessageHandler.log"
ShutDownLogFileNameTemplate	Applic/Processing/Logging	1	string		"GenericMessageHandler_shutdown.log"

There are three main sections in this document:

- Input
- Output
- Processing

Input

This section defines the input source and the way input messages will be handled.
There are currently three types of input handled: File, Queue and Socket (server).
Processing based on file input will be in batch mode; that is GeMHa will process the input files and immediately close down. When receiving messages from a Queue or Socket, GeMHa will block indefinitely on the inbound connection.

If the input type is **File**, you specify the source directory and file name in the *InputFile* sub-section. In fact, if you specify a file name template (based on the java.util.regex.Pattern rules), all files meeting the selection criteria will be processed in order (unless *SortFilteredFileNames* option is explicitly set to false).

If *DataFormatType* is “XML”, each file is assumed to contain one XML message, which will be processed individually.

If *DataFormatType* is “CSV”, each file is assumed to contain rows of character-separated values. Each row will be converted to an XML message and processed individually. You describe the format of the input file in the *CSVParams* element.

If the input type is **Queue**, in the *InputQueue* sub-section you specify the URL to the Messageing System server/broker and the name of the Queue from which you wish to receive requests. Setting the *MilliSecondsBeforeQuiet* value controls temporarily closing output media – if no messages are received *MilliSecondsBeforeQuiet* milliseconds after the last message, the output medium for processing and for responses are temporarily closed and we'll block indefinitely on the input Queue.

If the input type is **Socket**, you specify the number of the Port at which the socket server will listen for requests in the *InputSocket* sub-section.

Output

This section defines the response medium and the way response messages, returned by the processor, will be handled. There are currently three types of response output handled: File, Queue and HTTP post.

If the Message Processor (see later) is defined to return a response, indicating success or failure of the request, you define the medium to which these response messages will be sent within the *Output* element.

If the output type is **File**, you specify the destination directory and file name in the *FileNameTemplate* element, including the full path to the destination directory. The name of the file can be a simple name or can represent a template from which the file name will be created. This template allows you specify two 'placeholders' in the name that will be replaced by current information, these are: '*' in the file name will be replaced by the Audit Keys for the message, '?' will be replaced by a Date-time stamp plus a serial number (for uniqueness). For example, D:\myapp\responses\response_*_?.xml could become D:\myapp\responses\response_1025_201310041112331.xml

If the output type is **Queue**, in the *OuputQueue* sub-section you specify the URL to the Messageing System server/broker and the name of the Queue to which you wish to send responses.

If the output type is **HTTP**, in the *OuputHTTP* sub-section you specify the URL of the server hosting the web server and the name of the End Point within that web server that will handle the message. It is recommended that you do not set *HTTPWithBackoff* to false as this ensures the server is not overwhelmed by repeated connection requests when busy.

Processing

This section defines the target Message Processor that will handle the input messages and optionally return responses. The existence of the earlier *Output* section implies that the processor will return a response.

The *MessageProcessingClassName* element names the class that will perform the processing in this instance. Valid values are:

gemha.servers.ProcessMessageForDb
gemha.servers.ProcessMessageForSocket
gemha.servers.ProcessMessageForFile
gemha.servers.ProcessMessageDoNothing

Each processing class will in turn have its own *Configuration* file, defining further settings required for handling messages - see the individual sections later describing the processing for each of these options.

If a message is successfully processed, it will be consumed – for example removed from an Input Queue or acknowledged as successfully processed by the Socket server.

If a message produces an error, it will generally not be consumed – for example it would be left on an Input Queue or acknowledged as failed by the socket server. Certain error-processing behaviour can be defined in the *Application Configuration* file – see the description of the *InputValidation*, *Auditing* and *DataContractName* elements.

The following list describes each element in the Application Configuration File...

Tag Name	Description
Applic	Root of the settings document
Params	General application parameters.
ExternalShell	The shell used to issue native commands. For example to get the Process ID.
Input	
InputSource	Elements in this aggregate define the input medium
InputLimit	The number of messages to be read. No more messages will be read after this limit is reached.
DataFormat	XML or CSV
DataContractName	An attribute for this element called Location defines the path to an element in the incoming message containing the name of the contract being met. The value in DataContractName will be compared to that in the message. If they are found to mis-match (or it is missing in the message) the attribute ActionOnError will define what then happens: a value “shutdown” causes the application to terminate; “discard” skips the current messsage and carries on – the errant message is saved to a file whose name is determined by the *ErrorFileNameTemplate* tag in the *ErrorFiles* aggregate (see below).
InputQueue	Contains settings controling inputting from a message Queue
QueueManagerName	*deprecated*
QueueName	The name of the Queue from which to retrieve input messages.
MilliSecondsBeforeQuiet	Number of milliseconds we'll wait for a message before going quiet (temporarily closing down resources)
InputSocket	Contains settings controling inputting from a socket
PortNumber	The port on which a socket server will listen for requests.
InputFile	Contains settings controling inputting from a file or fileset
FileNameFilter	The pattern to use to identify the names of files containing input messages, where the pattern is based on the java.util.regex.Pattern rules.
FileDir	Directory in which to find input files. Default is “.” (the current working directory). If the supplied name does not end in a filename separator, one will be added.
SortFilteredFileNames	If true (the default), and *FileNameFilter* produces a list of files to be processed, this list will to be sorted before processing. In this way, if filenames included a datetime stamp, they could be ordered by age. If false, the order is unspecified.
CSVParams	Settings for use when the input is from a Character-separated, record-per-row file. Note: The *AcceptMessagesFromFiles* class creates an XML message from the records read and passes this on for processing. By default it will create one XML message per row, but you can change this behaviour with the *MaxRecsPerMessage* setting. The format of the newly-built XML message will be that of a DBAction (see *XMLFormat* below).
FieldSeparator	The character(s) that separate fileds in a row of input data. (If the value \t is supplied, it will be replaced by the actual tab character.)
MaxRecsPerMessage	Number of input records to include in each newly-built input message.
NumRecordsToSkip	Number of input records to skip when reading from file(s). These records will not be processed. This counter works across all files processed. It is useful when processing stopped at some point and we need to restart from where we left off.
ColumnOrder	Contains a list of names for the columns in the input file, in the order they appear in the file. Note these names do not actually appear in the input file itself, but will be used to name the XML tags in the newly-built input message.
Column	The name to be given to data in a column in the input file when that row is converted to an XML message. The order of these COLUMN elements represents the column order in the file.
XMLFormat	Indicates the shape of XML to be created from a read CSV row. There are currently two possible values: SELECT (the default) produces a message with the following shape: /MESSAGE/DBACTION/TABLE containing COLUMNS elements, in turn containing the COLUMN elements. INSERT produces a message with the following shape: /MESSAGE/DBACTION/INSERT containing COLUMNS elements, in turn containing the COLUMN elements – plus the settings in *InsertParams* aggregate (see below), if they exist. When this message is to indeed be processed against a database, the table name specified in the *DefaultTablename* element in the *ForDb* configuration file will be added to each DbAction.

InsertParams	If the value for the XMLFormat parameter is "insert", the tags within this aggregate may be used to further define the insert statement(s) to be created.
Action_On_Error	The action to be taken when the created INSERT fails on execution in the database. A value of “shutdown” causes the application to terminate; “discard” skips the current messsage and carries on – the errant message is saved to a file whose name is determined by the *ErrorFileNameTemplate* tag in the *ErrorFiles* aggregate (see below).
Prepared_Statement_Name	The name of a Prepared Statement to use with the created INSERT. If this is not specified, the table name specified in the *DefaultTablename* element in the *ForDb* configuration file will be added to each DbAction.
Immediate_Commit	The setting for whether the created action should be committed immediately after execution (or be committed along with other transactions). This option should NOT be set to Yes if the *AutoCommit* option in the *ForDb* configuration file is set to true – MySQL, for example, will throw an exception if we try to explicitly commit a transaction with Autocommit=true for the connection.
InputValidation	If exists, determines whether (and type of) Schema-based validation is to be applied to the input message.
Output	If this aggregate is missing, no message will be output.
OutputQueue	If outputting to a Queue, contains the Queue details.
QueueManagerName	*deprecated*
QueueName	The name of the Queue to which to send output messages.
ReplyToQueueManagerName	*deprecated*
ReplyToQueueName	The Queue name to stamp on output messages. This will be overridden by any ReplyToQueue name on an incoming message.
OutputFile	If outputting to a File, contains the File details.
FileNameTemplate	The template to use to construct a file name for each output message, including the path to the output directory, if required • * will be replaced by the Audit Keys for the message. • ? Will be replaced by a Date-time stamp plus a serial number (for uniqueness) Example: D:\myapp\responses\response_*_?.xml
OutputHTTP	If outputting over an HTTP connection, contains the connection details.
ServerUrl	The URL of the server hosting the web server, for example HTTP://localhost:8080
EndPointName	The name of the End Point residing on the server, for example the servlet name
HTTPWithBackoff	If true (default), when trying to connect to the server and we receive a 'busy' response, we will 'back off' in the following way: wait 2+r seconds (where r is a random number between 0 and 0.9) and try again; if we again get a 'busy' response we'll now wait twice as long as the last time; we will stop trying after 5 attempts and throw an exception, causing the application to close down.
Processing	Encapsulates processing instructions.
MessageProcessingClassName	The name of the class to load, which will handle/process messages. This class must implement the *IProcessMesssage* interface.
MessageProcessingSettingsFileName	The name (including path) of the *Configuration* file for the *MessageProcessingClassName* above, containing settings specific to this type of processing.
MinResponsesExpected	The minimum number of response messages expected from the class named in *MessageProcessingClassName* above. The message will fail if this restriction exists and is not met. *deprecated*
MindElements	Contains the collection of elements (aggregates and tags) from the input message to be retained while processing the message and returned (added back) in the output message.
ElementName	Name (and path) of an element of the input message to be retained and added back to the output message.
SendElements	Contains the collection of aggregates and tags from the input message to be sent in the message destined for the Message Processor class (see *MessageProcessingClassName* above).
ElementName	Name (and path) of an element of the input message to be sent in the message destined for the Message Processor class (see *MessageProcessingClassName* above).
TargetMainDocElementName	The name to be given to the Root element of the message sent to the Message Processor class (see

	MessageProcessingClassName above). If not specified, the name of the root element will be carried over from the input message.
ResponseMainDocElementName	The name to be given to the Root element of the output message. If specified, the message returned from the Processor will be wrapped in an element of this name. If not specified, the name of the oput message will be saved to medium "as is".
ResponseLiterals	Contains the collection of elements to be altered in/added to the output message.
ResponseLiteral	The value to assign a tag in the output message, with the name of the tag given in the Location attribute. If the tag does not exist in the output, it will be created.
Auditing	
AuditKeys	Contains the collection of tag pathnames to identify the Audit Key of an input message. This Audit Key uniquely identifies a message and will be used track messages in the log file (and may possibly be included in the filename, if the output is put to file).
KeyName	The path to the value in the input message which will form all or part of the Audit Key for that message.
AuditKeysSeparator	If this exists, the character(s) will be used to separate individual values within the Audit Key for a message, if more than one tag is used to build the key. Otherwise multiple values will just be concatenated.
ErrorFiles	
ErrorFilesDir	The directory to which to save any errant messages. Default is "." (the current working directory). If the supplied name does not end in a filename separator, one will be added.
ErrorFileNameTemplate	The template to use to build the filenames for errant messages. • * will be replaced by the Audit Keys for the message • ? Will be replaced by a Date-time stamp plus a serial number (for uniqueness) Example: response_*_?.xml Default: ErrorMessage_*_?.txt
Logging	
Level	If this aggregate is missing, the default Logging Level will be "CONFIG" (i.e Production Level)
EnvironmentLoggingLevel	Use this instead of *GeneralLoggingLevel* to choose Logging Level according to the environment in which the application will run. Valid values are Development (FINER), Test (FINE), Production (CONFIG).
GeneralLoggingLevel	Use this instead of *EnvironmentLoggingLevel* to choose an exact Logging Level. Valid values are those defined by the Java java.util.logging.Logger class... The levels in descending order are: • SEVERE (highest value) • WARNING • INFO • CONFIG • FINE • FINER • FINEST (lowest value) • In addition there is a level OFF that can be used to turn off logging, and a level ALL that can be used to enable logging of all messages.
LogFileDir	The directory in which log files will reside. Default is "." (the current working directory). If the supplied name does not end in a filename separator, one will be added.
LogFileNameTemplate	The template to use to build the Filename for the normal Log file. • * will be replaced by the value for MessageProcessingClassName above • $ will be replaced by the Process ID of this instance • ? Will be replaced by a Date-time stamp plus a serial number (for uniqueness)
ShutDownLogFileNameTemplate	The template to use to build the Filename for the shutdown-process Log file. For technical reasons related to Java Logging, Logging switches to this file during garceful shutdown, so the last few log messages will go to this file instead. • * will be replaced by the MessageProcessingClassName above • $ will be replaced by the Process ID of this instance • ? Will be replaced by a Date-time stamp plus a serial number (for uniqueness)

Description of Processors

DoNothing

This “dummy” handler simply returns the message presented to it, making no changes. It merely binds the input stream to the output stream.

This allows us send…

> files of messages
> or
> file(s) of CSV records (will be converted to XML messages internally)
> or
> messages from a Queue
> or
> messages sent to a Socket server

to…

> files of messages
> or
> a Queue
> or
> a Web application via HTTP

ForDB

This handler accepts XML-based instructions, conforming to a specific contract, to perform SELECTS, INSERTS, UPDATES and DELETES on a specified database connection.

The message must conform to the following:

```
<MESSAGE>
	<DBACTION>
		<action>
		</action>
	</DBACTION>
</MESSAGE>
```

where *action* is one of SELECT, INSERT, UPDATE or DELETE.

Within the *action* element, you specify a TABLENAME element and a COLUMNS and/or WHERE element, depending on the type of *action* to be performed. Actions may be multiple and mixed within a single message and will be performed in the following order: all INSERTS, followed by all UPDATES, followed by all DELETES, followed by all SELECTS.

You may add simple elements to the DBACTION aggregate and specify them (in the *Application Configuration* file) as an identifying key for the message, whose value(s) will be output as part of logging messages for auditing purposes (they can have any name you choose).

INSERTs

Here is an example of an INSERT request...

```
<MESSAGE>
	<DBACTION>
		<KEY>1002</KEY>
		<ACTION_ON_ERROR>RESPOND</ACTION_ON_ERROR>
		<INSERT>
			<TABLENAME>LW_delme1</TABLENAME>
			<COLUMNS>
				<USERID>1</USERID>
				<USERCODENAME>'098rety'</USERCODENAME>
				<USERSTATUS>'SUSPENDERS'</USERSTATUS>
				<ADMINCOMMENT>'lemmo1'</ADMINCOMMENT>
				<UPDATEDATETIME>2008-08-13</UPDATEDATETIME>
			</COLUMNS>
		</INSERT>
		<INSERT>
			<TABLENAME>LW_delme1</TABLENAME>
			<COLUMNS>
				<USERID>2</USERID>
				<USERCODENAME>'098rety'</USERCODENAME>
				<USERSTATUS>'SUSPENDERS'</USERSTATUS>
				<ADMINCOMMENT>'lemmo1'</ADMINCOMMENT>
				<UPDATEDATETIME>2008-08-20</UPDATEDATETIME>
			</COLUMNS>
		</INSERT>
```

```
        </DBACTION>
</MESSAGE>
```

The elements within the COLUMNS aggregate will match columns in the target table.
You would specify (in the *Configuration* file, see later) the MESSAGE/DBACTION/KEY element as containing an audit key. If you specfied 2 elements, their values would be concatenated to form the audit key, separated by a specified character (e.g. a “-”).

The ACTION_ON_ERROR element, if included, tells the processor what to do in the event of an error while performing the action. Here we want a response to be returned to sender, after which we'll carry on processing the next message. If you specified EXCEPTION, an exception would be thrown, stopping processing.

UPDATEs

Here is an example of an UPDATE request...

```
<MESSAGE>
        <DBACTION>
                <KEY>1002</KEY>
                <ACTION_ON_ERROR>RESPOND</ACTION_ON_ERROR>
                <UPDATE>
                        <TABLENAME>users.lw_delme1</TABLENAME>
                        <COLUMNS>
                                <USERSTATUS>'LOCKEDOUT'</USERSTATUS>
                                <ADMINCOMMENT>'updated status'</ADMINCOMMENT>
                        </COLUMNS>
                        <WHERE>
                                <USERID>1</USERID>
                        </WHERE>
                </UPDATE>
        </DBACTION>
</MESSAGE>
```

The elements within the COLUMNS aggregate will match columns in the target table to be updated. The elements within the WHERE aggregate will become equality operations in the SQL WHERE clause – for example, the above request would produce the following SQL statement:

```
UPDATE  users.lw_delme1
SET USERSTATUS = 'LOCKEDOUT', ADMINCOMMENT = 'updated status'
WHERE USERID = 1;
```

DELETEs

Here is an example of an DELETE request...

```
<MESSAGE>
        <DBACTION>
                <KEY>1002</KEY>
                <ACTION_ON_ERROR>RESPOND</ACTION_ON_ERROR>
                <DELETE>
                        <TABLENAME>users.lw_delme1</TABLENAME>
```

```
				<WHERE>
					<USERID>2</USERID>
					<NAME>'Charles Ingels'</NAME>
				</WHERE>
			</DELETE>
		</DBACTION>
</MESSAGE>
```

The elements within the WHERE aggregate will become equality operations in the SQL WHERE clause – for example, the above request would produce the following SQL statement:

```
DELETE FROM users.lw_delme1
WHERE USERID = 2
AND NAME = 'Charles Ingels';
```

SELECTs

Here is an example of an DELETE request...

```
<MESSAGE>
		<DBACTION>
			<KEY>1002</KEY>
			<ACTION_ON_ERROR>RESPOND</ACTION_ON_ERROR>
			<SELECT>
				<TABLENAME>LW_delme1</TABLENAME>
				<COLUMNS>
					<USERID></USERID>
					<USERCODENAME></USERCODENAME>
					<USERSTATUS></USERSTATUS>
					<ADMINCOMMENT></ADMINCOMMENT>
				</COLUMNS>
				<WHERE>
					<USERSTATUS>'ACTIVE'</USERSTATUS>
				</WHERE>
			</SELECT>
		</DBACTION>
</MESSAGE>
```

The elements within the COLUMNS aggregate will match columns in the target table to be included in the result. The elements within the WHERE aggregate will become equality operations in the SQL WHERE clause – for example, the above request would produce the following SQL statement:

```
SELECT USERID,
       USERCODENAME,
       USERSTATUS,
       ADMINCOMMENT
FROM users.lw_delme1
WHERE USERSTATUS = 'ACTIVE';
```

If an error occurs, a response message will be returned to the sender, explaining the problem. For example, if we failed to surround the USERSTATUS value of ACTIVE with single quotes, the SQL compiler (for MySQL) would complain...

```
<MESSAGE>
	<DBACTION>
	<SELECT>
		<TABLENAME>users.LW_delme1</TABLENAME>
		<WHERE>
			<USERSTATUS>'ACTIVE'</USERSTATUS>
		</WHERE>
		<STATUS>FAILED</STATUS>
		<NUM_SUCCESSFUL>0</NUM_SUCCESSFUL>
		<ERROR_CODE>1054</ERROR_CODE>
		<ERROR_TEXT>DbConnection.getQueryResults(): SELECT query failed.
			Caught exception: Unknown column 'ACTIVE' in 'where clause' Error
			Code: 1054 SQL State: 42S22
		</ERROR_TEXT>
	</SELECT>
	<KEY>1002</KEY>
	</DBACTION>
</MESSAGE>
```

A successful query would return something like this to the sender...

```
<MESSAGE>
	<DBACTION>
		<SELECT>
			<TABLENAME>users.LW_delme1</TABLENAME>
			<WHERE>
				<USERSTATUS>'ACTIVE'</USERSTATUS>
			</WHERE>
			<STATUS>EXECUTED</STATUS>
			<NUM_SUCCESSFUL>2</NUM_SUCCESSFUL>
			<TABLE>
				<ROW>
					<AdminComment>lemmo2</AdminComment>
					<UserId>2</UserId>
					<UserCodeName>098rety</UserCodeName>
					<UserStatus>ACTIVE</UserStatus>
				</ROW>
				<ROW>
					<AdminComment>me too</AdminComment>
					<UserId>10</UserId>
					<UserCodeName>78tt</UserCodeName>
					<UserStatus>ACTIVE</UserStatus>
				</ROW>
			</TABLE>
		</SELECT>
		<KEY>1002</KEY>
	</DBACTION>
</MESSAGE>
```

...these are the rows that matched the selection criteria of the SELECT request.

Prepared Statements

SELECT queries and data manipulation language (DML) queries, specified in the received XML message, are limited to simple, value-matching criteria. This is because all qualifications supplied in the XML are simply ANDed together when forming the WHERE clause of the SQL statement.

More complex requests, however, can be specified in the *Applications Configuration* file as *Prepared Statements*, with the incoming XML messsage then supplying expected parameters.

Here's an example of a request dependent on two different Prepared Statements...

```
<MESSAGE>
	<DBACTION>
		<KEY>1002</KEY>
		<ACTION_ON_ERROR>RESPOND</ACTION_ON_ERROR>
		<INSERT>
			<PREPARED_STATEMENT_NAME>
				LW_delme1_insert_001
			</PREPARED_STATEMENT_NAME>
			<COLUMNS>
				<USERID>1</USERID>
				<USERCODENAME>'098rety'</USERCODENAME>
				<USERSTATUS>'SUSPENDERS'</USERSTATUS>
				<ADMINCOMMENT>'lemmo1'</ADMINCOMMENT>
				<UPDATEDATETIME>2008-08-13</UPDATEDATETIME>
			</COLUMNS>
		</INSERT>
		<UPDATE>
			<PREPARED_STATEMENT_NAME>
				LW_delme1_update_001
			</PREPARED_STATEMENT_NAME>
			<COLUMNS>
				<USERID>1</USERID>
			</COLUMNS>
		</UPDATE>
	</DBACTION>
</MESSAGE>
```

The first *action* is an INSERT request which passes 5 parameter values to the prepared statement LW_delme_insert_001. This prepared statement will be populated with the values from the elements within the COLUMNS aggregate and executed.

The second *action* is an UPDATE request which passes a single parameter value to the prepared statement LW_delme_update_001. Again, this prepared statement will be populated with the value from the COLUMNS aggregate and executed.

You can also mix *actions* referring to tables with those referring to prepared statements.

ForDb Configuration File

The name and location of the *ForDb Configuration* file is specified in the

Applic/Processing/MessageProcessingSettingsFileName

element in the *Application Configuration* file.

This XML-format file contains additional configuration directing processing to a database. See the table on the next page for a description of the configuration settings...

The following list describes structure of the ForDb Configuration File...

Element Name	**Parent**	**Occurrence**	**Type**	**Domain /Format**	**Default**
Applic		1	Aggregate		
Params	Applic	1	Aggregate		
JdbcClass	Params	1	String		
DbURL	Params	1	String		
UserName	Params	1	String		
UserPass	Params	1	String		
AutoCommit	Params	0..1	String	on/off	on
DefaultTablename	Params	0..1	String		
PreparedStatements	Params	0..1	Aggregate		
PreparedStatement	PreparedStatements	0..n	Aggregate		
PreparedStatementName	PreparedStatements	1	String		
PreparedStatementSQL	PreparedStatements	1	String		
ParameterOrder	PreparedStatements	0..1	Aggregate		
Column	ParameterOrder	1..n	String		
Auditing	Applic	0..1	Aggregate		
AuditKeys	Applic/Auditing	0..4	Aggregate		
KeyName	Applic/Auditing/AuditKeys	1..n	string		
AuditKeysSeparator	Applic/Auditing	0..1	string		

The following list describes each element in the ForDb Configuration File...

Element Name	Description
Applic	Root of the configuration document
Params	General application parameters
JdbcClass	The name of the JDBC driver for the selected database
DbURL	The URL with which to connect to the database
UserName	The User Name with which to connect to the database
UserPass	The Password with which to connect to the database
AutoCommit	If on, all SQL statements submitted for the connection will be executed and **committed** as individual transactions
DefaultTablename	Name to use for database table, for use if none is supplied as part of a transaction
PreparedStatements	Contains all Prepared Statements
PreparedStatement	Parameterised SQL statement that can be called by messages, supplying the values for the parameters
PreparedStatementName	The unique identifier for the Prepared Statement
PreparedStatementSQL	The SQL that will be used to generate the Prepared Statement - this must be valid SQL. It may also be parameterised
ParameterOrder	For a parameterised Prepared Statement, holds the list of parameters for the Prepared Statement, in the order they appear in the PreparedStatementSQL
Column	A name of a parameter, as it is understood by the database, which will also match the name supplied in the incoming message.
Auditing	Contains all auditing settings
AuditKeys	Contains the collection of tag path names to identify the Audit Key of an individual Action (i.e of an Insert, Select etc.). This Audit Key uniquely identifies a message-action and will be used to track messages in the log file. There can be up to four of these key-sets and they are identified by the DbAction attribute, which will be one of: insert, update, delete or select.
KeyName	The path to the value in the input-message Action which will form all or part of the Audit Key for that message.
AuditKeysSeparator	If this exists, the character(s) will be used to separate individual values within the Audit Key for a message, if more than one tag is used to build the key. Otherwise multiple key values will simply be concatenated.

ForSocket

This is a handler for placing the XML message on a socket connection, for use when an application has no access to a messaging system, for example Websphere.

This module uses the services of the *Socket* project, which supplies both the client- and server-side wrapper classes for safely sending a message (as a string) across a Socket connection.
The Socket server will acknowledge receipt of the message.

If it is required, the Socket server can also return an acknowledgement that the message was processed successfully on that side of the connection.

The following list describes structure of the *ForSocket* Configuration File...

Element Name	Parent	Occurrence	Type	Domain /Format	Default
Applic		1	Aggregate		
Params	Applic	1	Aggregate		
PortNumber	Params	1	Positive Integer		
HostName	Params	1	String	localhost	
ApplicationLevelResponse	Params	1	ResponseType	asynchronous, synchronous	
Auditing	Applic	0..1	Aggregate		
AuditKeys	Applic/Auditing	1	Aggregate		
KeyName	Applic/Auditing/AuditKeys	1..n	string		
AuditKeysSeparator	Applic/Auditing	0..1	string		

The following list describes each element in the *ForSocket* Configuration File...

Element Name	Description
Applic	Root of the configuration document
Params	General application parameters
PortNumber	The port at which the socket server is waiting for connection requests
HostName	The URL of the server on which the socket server resides
ApplicationLevelResponse	If set to asynchronous, the message will be sent to the socket server, which will acknowledge receipt, but will not inform the client of the outcome of the request contained within the message. If set to asynchronous, after acknowledging receipt of the message, the socket server will return an application-level response to inform the client of the outcome of the request contained within the message.
Auditing	Contains all auditing settings
AuditKeys	Contains the collection of tag path names to identify the Audit Key of an individual request to the socket server. This Audit Key uniquely identifies a message request and will be used to track messages in the log file.
KeyName	The path to the value in the input-message Action which will form all or part of the Audit Key for that message.
AuditKeysSeparator	If this exists, the character(s) will be used to separate individual values within the Audit Key for a message, if more than one tag is used to build the key. Otherwise multiple key values will simply be concatenated.

ForFile

With this handler, XML messages can be output as rows of character-separated fields to a named file. You specify the Columns Location element (name and path of the aggregate that will contain the rows to be sent to the output file) and each occurrence of that element will be processed to form an output row.

Here's an example of a message that could be output to a file...

```
<MESSAGE>
	<FILE_REQUEST>
		<KEY>123456789G</KEY>
		<TABLE>
			<ROW>
				<COLUMNS>
					<USERID>1</USERID>
					<USERCODENAME>098rety</USERCODENAME>
					<USERSTATUS>SUSPENDERS</USERSTATUS>
					<ADMINCOMMENT>lemmo1</ADMINCOMMENT>
					<UPDATEDATETIME>13-08-2008</UPDATEDATETIME>
				</COLUMNS>
			</ROW>
			<ROW>
				<COLUMNS>
					<USERID>2</USERID>
					<USERCODENAME>098rety</USERCODENAME>
					<USERSTATUS>SUSPENDERS</USERSTATUS>
					<ADMINCOMMENT>lemmo1</ADMINCOMMENT>
					<UPDATEDATETIME>13-08-2008</UPDATEDATETIME>
				</COLUMNS>
			</ROW>
		</TABLE>
	</FILE_REQUEST>
</MESSAGE>
```

So, if you specify `/MESSAGE/FILE_REQUEST/TABLE/ROW/COLUMNS` as the path to the columns aggregate, 2 rows of 5 columns of data wil be output. Note that the names of elements in the input message can be any value and the "shape" of the message is not fixed – it need only make sense in terms of extracting column name/value pairs from the message.

Another setting in the *ForFile* configuration file, *IncludeColumnNames*, controls whether column names are also included as the first row of the output file.

The following list describes structure of the *ForFile* Configuration File...

Element Name	Parent	Occurrence	Type	Domain /Format	Default
Applic		1	Aggregate		
Params	Applic	1	Aggregate		
ColumnsLocation	Params	1	String		/MESSAGE/FILE_REQUEST/TABLE/(ROW/COLUMNS)
MessagesFileNameTemplate	Params	1	String		ProcessMessageForFile_?.txt
FieldSeparator	Params	0..1	String		\t
IncludeColumnNames	Params	0..1	Boolean	true/false	TRUE
FileOpenMode	Params	0..1	FileOpenModeType	create, append	create
Auditing	Applic	0..1	Aggregate		
AuditKeys	Applic/Auditing	1	Aggregate		
KeyName	Applic/Auditing/AuditKeys	1..n	string		
AuditKeysSeparator	Applic/Auditing	0..1	string		

The following list describes each element in the *ForFile* Configuration File...

Element Name	Description
Applic	Root of the configuration document
Params	General application parameters
ColumnsLocation	The path to the aggregates in the incoming message that will contain the column data. The 'xml-text-node' value of any direct child elements of this aggregate will be output to the file. Note: normally the final node in the location path should be the repeating element. For example, in the following `/MESSAGE/FILE_REQUEST/TABLE/ROWS` the ROWS element should be repeating, while all before it should only occur once. If, however, it is the second-last element that repeats, you can surround the final two elements in brackets. For example, in the following `/MESSAGE/FILE_REQUEST/TABLE/(ROW/COLUMNS)` the ROW element repeats and each ROW contains only one COLUMNS aggregate.
MessagesFileNameTemplate	The template to use to build the filename for the output file. • * will be replaced by the Audit Keys for the message • ? Will be replaced by a Date-time stamp plus a serial number (for uniqueness) Example: Expenses_output_*_?.xml Default: ProcessMessageForFile_?.txt
FieldSeparator	The character(s) that separate(s) fileds in a row of output data. (If the value \t is supplied, it will be replaced by the actual tab character.)
IncludeColumnNames	If true, the first row output will be the list of column names (noted from the input message), separated by the *FieldSeparator.*
FileOpenMode	create=> open a new file, append=> open the file for appending (the default is *create*)
Auditing	Contains all auditing settings
AuditKeys	Contains the collection of tag path names to identify the Audit Key of an individual request to the socket server. This Audit Key uniquely identifies a message request and will be used to track messages in the log file.
KeyName	The path to the value in the input-message Action which will form all or part of the Audit Key for that message.
AuditKeysSeparator	If this exists, the character(s) will be used to separate individual values within the Audit Key for a message, if more than one tag is used to build the key. Otherwise multiple key values will simply be concatenated.

General Features

Logging

Java Logging of all events, exceptions and errors is used throughout all modules, allowing selection of the level of messages output to the log file, according to the seven levels of the Java Logging specification. The level is set in the *Logging* section of the settings files.

Audit Trail

All modules support specification of the location of audit keys within XML messages. These keys are then included in output to the log file, when recording significant events and message-based exceptions. A trail is therefore available in the log file, showing when/if the message has been processed, consumed and responded to.

XML Validation

The input XML message can be defined to be validated by a specified XML Schema Definition file. If an error occurs, a parameter of the settings file will determine what should occur in the event of an error.

XML Validation error processing can be set to one of three options:

- Discard the message to an error file, continuing with the next message
- Shut down the application, leaving the message un-processed (for example, it remains on the input Queue)
- Respond to Sender with an error message, consuming the original message

Resource Management

When appropriate, all modules will close any resources when the Message Handler is accepting messages from a Queue or Socket and is determined to be in a state of “not busy”. This occurs when messages have not been received for a (parameter-driven) number of milliseconds.

Examples

Example 1 – Database Query

This accepts a database query request from a file, submits it to the database and returns the response (in this case a result set) to another file. Because input is file-based, the application runs in batch mode. It would be a simple matter to replace the Input and Output settings to be based on Queues.

Application Configuration File (Applic_FtoDBtoF_001.xml)

```
<Applic  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Applic.xsd">
      <Params>
            <ExternalShell>sh</ExternalShell>
      </Params>
      <Input>
            <InputSource>
                  <DataFormat>XML</DataFormat>
                  <InputFile>
                        <FileDir>C:\Users\wadel\git\GeMHa\GeMHa\data\</FileDir>
                        <FileNameFilter>select_001.xml</FileNameFilter>
                  </InputFile>
            </InputSource>
      </Input>
      <Output>
            <OutputFile>
                  <FileNameTemplate>
                  C:\Users\wadel\git\GeMHa\GeMHa\responses\response_*_?.xml
                  </FileNameTemplate>
            </OutputFile>
      </Output>
      <Processing>
            <MessageProcessingClassName>
                  gemha.servers.ProcessMessageForDb
            </MessageProcessingClassName>
            <MessageProcessingSettingsFileName>
C:\Users\wadel\git\GeMHa\GeMHa\ApplicSettingsFiles\ProcessMessageForDbSettings.xml
            </MessageProcessingSettingsFileName>
            <MindElements>
                  <ElementName>/MESSAGE/DBACTION/KEY</ElementName>
            </MindElements>
            <SendElements>
                  <ElementName>/MESSAGE/DBACTION</ElementName>
            </SendElements>
      </Processing>
      <Auditing>
            <AuditKeys ActionOnError="shutdown">
                  <KeyName>/MESSAGE/DBACTION/KEY</KeyName>
            </AuditKeys>
            <AuditKeysSeparator>-</AuditKeysSeparator>
            <ErrorFiles>
```

```
		<ErrorFilesDir>C:\Users\wadel\git\GeMHa\GeMHa\errors\</ErrorFilesDir>
				<ErrorFileNameTemplate>ErrorMessage_*_?.xml</ErrorFileNameTemplate>
			</ErrorFiles>
		</Auditing>
		<Logging>
			<Level>
				<GeneralLoggingLevel>FINEST</GeneralLoggingLevel>
			</Level>
			<LogFileDir>C:\Users\wadel\git\GeMHa\GeMHa\logs\</LogFileDir>
			<LogFileNameTemplate>GenericMessageHandler.log</LogFileNameTemplate>
			<ShutDownLogFileNameTemplate>
				GenericMessageHandler_shutdown.log
			</ShutDownLogFileNameTemplate>
		</Logging>
	</Applic>
```

ForDb Configuration File (ProcessMessageForDbSettings.xml)

```
<Applic  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ProcessMessageForDbSettings.xsd">
		<Params>
			<JdbcClass>com.mysql.jdbc.Driver</JdbcClass>
			<DbURL>jdbc:mysql://localhost:3306/users</DbURL>
			<UserName>corpUser01</UserName>
			<UserPass>dopple66ganger</UserPass>

			<AutoCommit>off</AutoCommit>
			<DefaultTablename>users.lw_delme1</DefaultTablename>
			<UpdateLockingStrategy>none</UpdateLockingStrategy>

		</Params>
		<PreparedStatements>
			<PreparedStatement>
				<PreparedStatementName>LW_delme1_insert_001</PreparedStatementName>
				<PreparedStatementSQL>
INSERT INTO users.lw_delme1
(ADMINCOMMENT,USERCODENAME,USERID,USERSTATUS,UPDATEDATETIME) VALUES (?,?,?,?,?)
				</PreparedStatementSQL>
				<ParameterOrder>
					<Column>ADMINCOMMENT</Column>
					<Column>USERCODENAME</Column>
					<Column>USERID</Column>
					<Column>USERSTATUS</Column>
					<Column>UPDATEDATETIME</Column>
				</ParameterOrder>
			</PreparedStatement>
			<PreparedStatement>
				<PreparedStatementName>LW_delme1_update_001</PreparedStatementName>
				<PreparedStatementSQL>
UPDATE users.lw_delme1 SET USERSTATUS = 'ACTIVE' WHERE USERID = ?
				</PreparedStatementSQL>
				<ParameterOrder>
					<Column>USERID</Column>
				</ParameterOrder>
			</PreparedStatement>
```

```
            <PreparedStatement>
                  <PreparedStatementName>LW_delme1_select_001</PreparedStatementName>
                  <PreparedStatementSQL>
SELECT * FROM users.lw_delme1 WHERE USERSTATUS = ?
                  </PreparedStatementSQL>
                  <ParameterOrder>
                        <Column>USERSTATUS</Column>
                  </ParameterOrder>
            </PreparedStatement>
<!--   This is an Oracle-specific type query
            <PreparedStatement>
                  <PreparedStatementName>LW_delme1_select_002</PreparedStatementName>
                  <PreparedStatementSQL ReturnType="XML">SELECT
XMLElement("USER",XMLAttributes(userid), XMLElement("DETAILS",
XMLForest(userCodeName,userStatus,adminComment))).getStringVal() AS theXML FROM
LW_delme1 WHERE USERSTATUS = ?</PreparedStatementSQL>
                  <ParameterOrder>
                        <Column>USERSTATUS</Column>
                  </ParameterOrder>
            </PreparedStatement>
 -->
      </PreparedStatements>
      <Auditing>
            <AuditKeys DbAction="insert">
                  <KeyName>COLUMNS/USERID</KeyName>
                  <KeyName>COLUMNS/USERCODENAME</KeyName>
            </AuditKeys>
            <AuditKeys DbAction="update">
                  <KeyName>WHERE/USERID</KeyName>
                  <KeyName>WHERE/USERCODENAME</KeyName>
            </AuditKeys>
            <AuditKeys DbAction="delete">
                  <KeyName>WHERE/USERID</KeyName>
                  <KeyName>WHERE/USERCODENAME</KeyName>
            </AuditKeys>
            <AuditKeys DbAction="select">
                  <KeyName>WHERE/USERSTATUS</KeyName>
            </AuditKeys>
            <AuditKeysSeparator>-</AuditKeysSeparator>
      </Auditing>
</Applic>
```

Input Data (select_001.xml)

```
<MESSAGE>
      <DBACTION>
            <KEY>1002</KEY>
            <ACTION_ON_ERROR>RESPOND</ACTION_ON_ERROR>
            <SELECT>
                  <TABLENAME>users.LW_delme1</TABLENAME>
                  <COLUMNS>
                        <USERID></USERID>
                        <USERCODENAME></USERCODENAME>
                        <USERSTATUS></USERSTATUS>
                        <ADMINCOMMENT></ADMINCOMMENT>
                  </COLUMNS>
```

```
                <WHERE>
                        <USERSTATUS>'ACTIVE'</USERSTATUS>
                </WHERE>
        </SELECT>
    </DBACTION>
</MESSAGE>
```

Execution

On the command line, from the GeMHa directory...

java -server -jar GeMHa.jar Applic_FtoDBtoF_001.xml

On the command line, from the GeMHa directory, via Maven...

mvn exec:java -Dexec.mainClass="gemha.GemhaBatch"
-Dexec.args="Applic_FtoDBtoF_001.xml"

Response Data (response_1002_201310031133250.xml)

```
<MESSAGE>
        <DBACTION>
                <SELECT>
                        <TABLENAME>users.LW_delme1</TABLENAME>
                        <WHERE>
                                <USERSTATUS>'ACTIVE'</USERSTATUS>
                        </WHERE>
                        <STATUS>EXECUTED</STATUS>
                        <NUM_SUCCESSFUL>2</NUM_SUCCESSFUL>
                        <TABLE>
                                <ROW>
                                        <AdminComment>lemmo2</AdminComment>
                                        <UserId>2</UserId>
                                        <UserCodeName>098rety</UserCodeName>
                                        <UserStatus>ACTIVE</UserStatus>
                                </ROW>
                                <ROW>
                                        <AdminComment>me too</AdminComment>
                                        <UserId>10</UserId>
                                        <UserCodeName>78tt</UserCodeName>
                                        <UserStatus>ACTIVE</UserStatus>
                                </ROW>
                        </TABLE>
                </SELECT>
                <KEY>1002</KEY>
        </DBACTION>
</MESSAGE>
```

Example 2 – Database Inserts

This accepts database INSERT requests from a CSV file, builds an XML message for every two rows, submits each INSERT to the database and returns a response for each pair to an individual file. (Setting *MaxRecsPerMessage* to 1 [default], will produce an XML message for each row and a corresponding response file.)

Application Configuration File (Applic_CtoDBtoF.xml)

```
<Applic  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Applic.xsd">
      <Params>
            <ExternalShell>sh</ExternalShell>
      </Params>
      <Input>
            <InputSource>
                  <InputLimit>7</InputLimit>
                  <DataFormat>CSV</DataFormat>
                  <InputFile>
                        <FileDir>data/</FileDir>
                        <FileNameFilter>input_CSV_[0-9]*.txt</FileNameFilter>
                        <CSVParams>
                              <FieldSeparator>\t</FieldSeparator>
                              <MaxRecsPerMessage>2</MaxRecsPerMessage>
                              <ColumnOrder>
                                    <Column>USERID</Column>
                                    <Column>USERCODENAME</Column>
                                    <Column>USERSTATUS</Column>
                                    <Column>ADMINCOMMENT</Column>
                                    <Column>UPDATEDATETIME</Column>
                              </ColumnOrder>
                              <XMLFormat>insert</XMLFormat>
                              <InsertParams>
                                    <Action_On_Error>RESPOND</Action_On_Error>
                                    <Immediate_Commit>YES</Immediate_Commit>
                              </InsertParams>
                        </CSVParams>
                  </InputFile>
            </InputSource>
      </Input>
      <Output>
            <OutputFile>
                  <FileNameTemplate>
                        C:\Users\wadel\git\GeMHa\GeMHa\responses\response_*_?.xml
                  </FileNameTemplate>
            </OutputFile>
      </Output>
      <Processing>
            <MessageProcessingClassName>
                  gemha.servers.ProcessMessageForDb
            </MessageProcessingClassName>
            <MessageProcessingSettingsFileName>
C:\Users\wadel\git\GeMHa\GeMHa\ApplicSettingsFiles\ProcessMessageForDbSettings.xml
            </MessageProcessingSettingsFileName>
            <SendElements>
                  <ElementName>/MESSAGE/DBACTION</ElementName>
            </SendElements>
      </Processing>
      <Auditing>
            <AuditKeys ActionOnError="shutdown">
                  <KeyName>/MESSAGE/DBACTION/INSERT/COLUMNS/USERID</KeyName>
            </AuditKeys>
            <AuditKeysSeparator>-</AuditKeysSeparator>
            <ErrorFiles>
                  <ErrorFilesDir>C:\Users\wadel\git\GeMHa\GeMHa\errors\
                  </ErrorFilesDir>
                  <ErrorFileNameTemplate>ErrorMessage_*_?.xml</ErrorFileNameTemplate>
            </ErrorFiles>
      </Auditing>
      <Logging>
            <Level>
                  <GeneralLoggingLevel>FINEST</GeneralLoggingLevel>
            </Level>
            <LogFileDir>C:\Users\wadel\git\GeMHa\GeMHa\logs\</LogFileDir>
            <LogFileNameTemplate>GenericMessageHandler.log</LogFileNameTemplate>
            <ShutDownLogFileNameTemplate>
                  GenericMessageHandler_shutdown.log
            </ShutDownLogFileNameTemplate>
      </Logging>
</Applic>
```

ForDb Configuration File (ProcessMessageForDbSettings.xml)

Same as in Example 1 – Database Query.

Input Data (input_CSV_001.txt, input_CSV_002.txt)

input_CSV_001.txt

```
1      '097rety'     'SUSPENDERS' 'lemmo1'     '2008-08-13'
2      '098rety'     'SUSPENDERS' 'lemmo2'     '2008-08-14'
3      'dsfsd8'      'SUSPENDERS' 'lemmo3'     '2008-08-15'
```

input_CSV_001.txt

```
4      'rt767w'      'ACTIVE'     'somecomment'      '2008-12-18'
5      'erttr'       'DEAD' 'acomment'    '2008-12-19'
6      'easdr'       'ALIVE'      'xcomment'   '2008-12-20'
```

Execution

On the command line, from the GeMHa directory...

java -server -jar GeMHa.jar Applic_CtoDBtoF.xml

On the command line, from the GeMHa directory, via Maven...

```
mvn exec:java -Dexec.mainClass="gemha.GemhaBatch"
-Dexec.args="Applic_FtoDBtoF_001.xml"
```

Response Data

response_1_201310041638580.xml

```
<MESSAGE>
	<DBACTION>
		<INSERT>
			<TABLENAME>users.lw_delme1</TABLENAME>
			<COLUMNS>
				<USERID>1</USERID>
				<USERCODENAME>'097rety'</USERCODENAME>
			</COLUMNS>
			<STATUS>COMMITTED</STATUS>
			<NUM_SUCCESSFUL>1</NUM_SUCCESSFUL>
		</INSERT>
		<INSERT>
			<TABLENAME>users.lw_delme1</TABLENAME>
			<COLUMNS>
				<USERID>2</USERID>
				<USERCODENAME>'098rety'</USERCODENAME>
			</COLUMNS>
			<STATUS>COMMITTED</STATUS>
			<NUM_SUCCESSFUL>1</NUM_SUCCESSFUL>
		</INSERT>
	</DBACTION>
</MESSAGE>
```

response_3_201310041638581.xml

```
<MESSAGE>
	<DBACTION>
		<INSERT>
			<TABLENAME>users.lw_delme1</TABLENAME>
			<COLUMNS>
				<USERID>3</USERID>
				<USERCODENAME>'dsfsd8'</USERCODENAME>
			</COLUMNS>
			<STATUS>COMMITTED</STATUS>
			<NUM_SUCCESSFUL>1</NUM_SUCCESSFUL>
		</INSERT>
		<INSERT>
			<TABLENAME>users.lw_delme1</TABLENAME>
			<COLUMNS>
				<USERID>4</USERID>
				<USERCODENAME>'rt767w'</USERCODENAME>
			</COLUMNS>
			<STATUS>COMMITTED</STATUS>
			<NUM_SUCCESSFUL>1</NUM_SUCCESSFUL>
		</INSERT>
	</DBACTION>
</MESSAGE>
```

response_5_201310041638582.xml

```
<MESSAGE>
       <DBACTION>
              <INSERT>
                     <TABLENAME>users.lw_delme1</TABLENAME>
                     <COLUMNS>
                            <USERID>5</USERID>
                            <USERCODENAME>'erttr'</USERCODENAME>
                     </COLUMNS>
                     <STATUS>COMMITTED</STATUS>
                     <NUM_SUCCESSFUL>1</NUM_SUCCESSFUL>
              </INSERT>
              <INSERT>
                     <TABLENAME>users.lw_delme1</TABLENAME>
                     <COLUMNS>
                            <USERID>6</USERID>
                            <USERCODENAME>'easdr'</USERCODENAME>
                     </COLUMNS>
                     <STATUS>COMMITTED</STATUS>
                     <NUM_SUCCESSFUL>1</NUM_SUCCESSFUL>
              </INSERT>
       </DBACTION>
</MESSAGE>
```