IBM®

Enterprise Exception Handling
Implementation Guide

Arunava Majumdar
Sr. IT Specialist
Focused Technology Practice
(Connectivity/BPM)
IBM Corporation.

Vineet Gupta
IBM Certified IT Specialist
Application Integration and Middleware
IBM Software Services for Websphere
IBM India Pvt. Ltd.

WebSphere. software

Contents

Table of Figures

Modification History

Date	Version	Author	Description
06/23/2009	0.2.0	Arunava Majumdar	Final release for Support Pac id08
05/01/2012	1.0.0	Arunava Majumdar, Vineet Gupta	Final release for Support Pac id08

Legal Disclaimer:

COPYRIGHT LICENSE:

In memory of my sister Anusree Majumdar

Acknowledgement:

Guy Hochstetler and I came up with the idea of a centralized Exception handling system for tracking down errors and cataloging them across the enterprise. This would help in troubleshooting very large and complex enterprise systems easily. The first objective was to standardize the exceptions generated from applications across the enterprise so that they can be analyzed that forms the basis for further developments on the idea. My special thanks to Guy for help cultivate the idea. The pattern since then had been successfully implemented at multiple organizations.

I also acknowledge the Open Source contributions on the following packages used in the delivery of the product.

Scope of the Document:

The scope of the document is limited to the design and implementation of the Enterprise Exception Handler. It provides the software specifications for the integration of the design with all applications in the organization.

The scope of the document is not to address how applications can handle exceptions within their code but how they can report exceptions into this common framework. It also addresses the present tools and future tools to enrich the exception detection and troubleshooting. It is left to the individual applications to address their own System and Business exception and numerous books and papers had been written on the subject.

This version of the paper provides a Java implementation of the design and API that may be incorporated in any Java application. The API provides methods to send the exception to an IBM WebSphere MQ® queue as a MQ message or a JMS message or to any generic JMS provider. Refer to javadocs for how to use the API.

Other API's may be developed for other languages based on the specifications provided in the document. C++ API will be provided at a later release.

Databases based on the schema and specifications provided in the document may be created for storing the exceptions in a consistent manner and for future exception analysis tools.

Integration with monitoring products with IBM Tivoli Omegamon™ will be provided as a future release.

What's new:

What's new in Enterprise Exception Handling Pattern version 1.0.0:

- ❖ Message expiry and the expired queue can be set for the Exception Catalog
- ❖ A description for the Exception Catalog is added to the schema
- ❖ Exception Catalog may be property file based where XML parsing is an issue
- ❖ Database table prefix may be provided (schema, catalog, etc.) to prefix table names
- ❖ Centralized logging
- ❖ Static linking of exception catalog using getExcCatInfo() generated from excutil
- ❖ Static loading of exception catalogs using EEH_SETUP environment variable
- ❖ EEH debug mode may be set from the environment variable , EEH_DEBUG
- ❖ Catalog registration with the EEH installation provides centralized management
- ❖ Catalog setup options added – setupRegisteredCatalogs(), setupCatalogs()
- ❖ AppId = "*" may be set in the Exception Catalog when used by multiple applications
- ❖ setAppId() API added for individual applications to set their id
- ❖ FileNotifier plugin sample added for exchd INotifier plug-in
- ❖ EEH Editor, an eclipse-based plug-in is provider for configuration, administration and analysis and re-queuing of exception data
- ❖ EEH Web Analyzer provides functionality to browse exceptions in the database and re-queuing of exception data

1. Introduction to Exceptions:

Exception Handling had always been an important topic of discussion while designing and architecting any operational system. Exceptions and failures will be present in any operational system and it is the intention of a robust design not just to minimize the exceptions but to handle the anomalies with the least effect to the operational system. Isolation, reporting, notification, monitoring and analysis of exceptions are hence, requirements for any exception handling system. A good exception handling process is essential to the hardening of the product whether it be mechanical, electrical, structural or software. This paper is limited to discussions on Software Exception Processing across the Enterprise in a controlled and standardized manner.

1.1. *What is an exception?*

An exception is a software runtime anomaly.

This is a very generic definition of an exception and the paper suggests a common exception handling process for the whole organization. All Application exceptions may be fed into the Enterprise Exception Handling Process in a standard manner for analysis and debugging. Exceptions for products are usually reported by monitoring products like IBM Tivoli Omegamon™. The Common Exception Handling process takes advantage of the monitoring infrastructure to report application exceptions.

1.2. *Exception Object concepts*

Errors detected inside the code are handled in different manners inside the code. A conditional statement may detect a failure of a certain nature and then decide to jump to a different location in the code. Jumping from one part of the code to another may become tedious to detect. While that may be the only option in certain languages like Assembly, calling functions or breaking from loops or just returning from the function is a better approach in structured languages like C or Basic. Object Oriented Programming introduced the concept of trying a piece of algorithm and catching exceptions at the end of it to handle these exceptions.

```
try {
        // Code to be tried
}
catch (Exception e) {
        // Exceptions caught and handled
}
```

1.3. *Importance of exception handling*

The robustness of an application depends on how well exceptions are being handled inside the code. Providing functionality and testing of the functionality is essential to meet the requirements, but hardening of the applications is necessary for any production-ready application. A well-hardened application should be able to trap all of runtime exceptions and reporting them in an efficient manner. While capturing all exceptions under every condition is not possible at the first deployment phase, the application must provide means of handling poison data sent to it for further analysis in the future. Subsequent analysis and additional exception handling inside the applications will elevate the maturity of the application and eliminate failures.

1.4. Importance of common exception handling for the organization

Every application may handle exceptions in the organization in a unique manner hardened and matured over a period of time. The handling of exceptions by individual applications may apparently stabilize the environment but the lack of an integrated view of the exceptions occurring throughout the organization may provide a false pretense of stability. It is very difficult and extremely resource intense to check individual log files for the any failures in the systems for each of the applications, especially in a medium to large size organization. Traceability and re-introduction of the data that caused the exception or the data that could not get processed due to the exception is often cumbersome. A complete picture of all the exceptions occurring in the system is therefore essential for the detection and analysis of the errors. A statistical analysis of these errors may also be provided at the end of the month or year to determine defects and improve the infrastructure and applications running in the organization.

1.5. Importance of cataloging exceptions

In order to achieve a common exception handling process in the organization, the first step would be to standardize and catalog all the exceptions that are presently handled by the applications and other system monitoring components. Each exception must be assigned a code and a set of parameters associated with it that fully qualifies the exception condition. Care must be taken to assign exception codes so that there is no duplication of the same type of exception with few different codes. Duplication of error codes from individual products is not the intent of the system either. E.g., it is not a good idea to assign an exception code to every SQLCODE or MQ REASONCODE but to group them under some standard problems they address, viz. connectivity, integrity, etc. However, products supporting the Enterprise Exception Handling standards and methodology may be directly incorporated in the exception catalog if they provide their own catalog. Each application generating these exceptions must be cataloged to determine what exceptions they are entitled to. Managing the catalog is important to derive accurate statistics of the exceptions occurring throughout the organization.

1.6. Designing exceptions

Designing good exceptions is the basis of a successful Exception Handling system. The exceptions must be explicit enough for quick problem determination as well as generic enough so as not to have an unnecessarily large exception catalog. The ability to preserve the dump at the point of exception (e.g. java Exception object, stack trace, ExceptionList inside the Message Broker Flow, etc.) in the exception message saves vital information for problem determination. Huge memory dumps are not often the most efficient way to troubleshoot a problem and should be avoided.

2. Enterprise Exception Handling Concepts:

Having emphasized on the importance on Exception Handling in applications let us consider how some of the ideas may be implemented in an organization and the steps required to adopt the Enterprise Exception Handling pattern.

2.1. Standardization of the Exception format

Exception Handling inside large organizations is mostly application centric and hence in most situations there exists no standardization of the structure of the exceptions. The foremost factor to enable exceptions to make sense at an organizational level is to have a standard for exceptions reported from any application. Taking it a step further, if a universal standard exits across organizations, then common tools may be developed for exception detection and troubleshooting purposes. The paper attempts to define a standard for exceptions after inputs were taken from multiple organizations on the requirements specific to that organization.

Standard API's provide a simple integration method for the EEH. Applications that had been already written may now use the API to format the exception messages without having to spend a lot of cycles in the integration effort. Concepts like the early detection of exceptions and handling wrongly formatted or generation of wrong exceptions should be caught at a development or testing level of the individual applications rather than at runtime. This is to prevent wrong exceptions giving rise to faulty statistical information.

Database provides a persistent searchable storage for exceptions in the architecture and such the schema for the EEH should also be standardized. The paper provides a schema for the implementation of the pattern after considering requirements from various organizations. Exceptions occurring in the Exception Handling system are also reported in the exception database in a consistent manner with specific exception category codes making the debugging of these special exceptions easy. Most likely these are only generated when applications have generated exceptions while generating exceptions and should be fixed as bugs in the application.

API standardization checks for exception parameters but the application header parameters are only checked at a database level (referential integrity).

2.2. Exception reporting through the ESB

Exceptions must be reported through a common information transport mechanism that is robust and resilient to network problems. The infrastructure must support multiple platforms and networks so that applications may report exceptions from any parts of a large heterogeneous organization. The mechanism must be asynchronous since the exception reporting system should not significantly affect the normal processing in the organization. Applications should be able to report exceptions and carry on with the normal functionality rather than synchronously waiting for a response. The Enterprise Service Bus (ESB) fits right into this architectural requirement. The ESB must provide guaranteed persistent delivery mechanism.

2.3. Notification through the Monitoring System

Notification mechanism must be provided and has be to extensible to integrate with monitoring systems so that the same monitoring mechanism may be used notify the Operations department of potential errors in the system.

Reporting errors through the monitoring system has its set of advantages. It provides an efficient and common process of reporting exceptions similar to the monitoring agents, e.g. Middleware or Database monitoring agents, to a centralized repository. The exception conditions may be co-related with other exceptions occurring in the system and being similarly reported to the monitoring sub-system, e.g. application database connections exceptions with a database crash and a situation may be build to shutdown the application. Another advantage is to be able to reset some exceptions when the problem has been fixed, e.g. a data integrity issue may be fixed by either manually or automatically (based on some rules) changing the data and re-introducing the data in the system at the point of failure and a notification automatically sent to reset the exception condition. In other words, closed-looped autonomic processes may be build around it.

2.4. Centralization and Statistical Analysis of Exceptions

The idea behind the Enterprise Exception Handing Pattern is to be able to centralize all application-related exceptions throughout the organization so that they can be analyzed with relative ease and help in the troubleshooting process. An extension of the same idea lies in the Statistical Analysis of exceptions over a period of time. As exceptions get reported centrally, analysis may be drawn from the category of exceptions received by every application in the organization or on which applications generate the most amount of exceptions. Some of these exceptions may be inevitable and the organization may have no control over them. Others might have inherent problems in the application or the interface to applications within the organization or partners. This provides a good basis for analyzing specific applications for these exceptions and enhancing the overall application performance in the long run. The reports may also be taking back to the customers who are not adhering to the standards of the application service interfaces causing discarding of data that could have been otherwise used. This happens especially in multinational organizations that interface with a large number of partners across the world.

2.5. Testing Exceptions Conditions

Exception Conditions are often overlooked by testing teams in the light of testing the functionality of the application. No less emphasis should be given to testing exception conditions if a robust application is desired. The ability for an application to report all runtime exceptions without causing a failure and reporting them correctly proves the resilience and the efficiency of the application in troubleshooting in a production environment.

The testing team must, therefore, determine all the exception conditions in the application and add them to the test cases. Every exception is properly cataloged and linked to the application catalog. Any application reporting exceptions that had not been associated with the application should be reported as a test case failure as well as exceptions with wrong parameters. To facilitate this process, every application must have its subset of the exception catalog and check against it. The testing team must make certain that are occurs at least one test case to test each of the exception codes that are assigned to the application. Most often multiple test cases with the same exception code will be present.

3. Overview of Enterprise Exception Handling

The Enterprise Exception Handling is a common framework to report exceptions in a consistent manner that is persisted in a database and analyzed later.

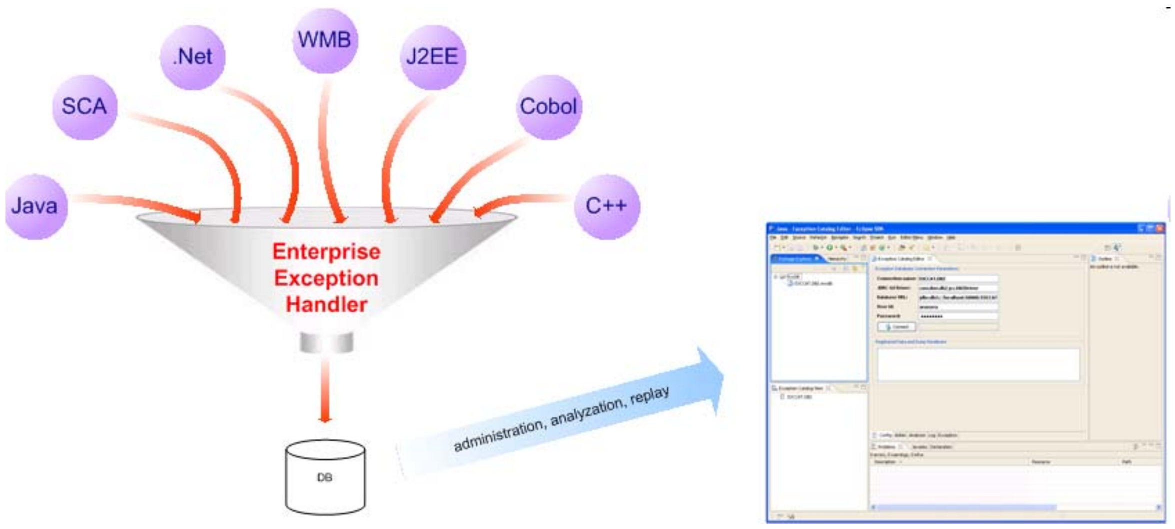

WMB
.Net
J2EE
SCA
Cobol
Java
C++
Enterprise Exception Handler
DB
administration, analyzation, replay

Figure 1 - EEH Overview

Based on the implementation of EEH, the process starts with the cataloging of exceptions while building an application. As the application encounters new exceptions to be handled they are noted and the exception catalog owner decides whether to add a new exception to the catalog or to assign an existing one to the application. Based on the exception to be used in the particular application, the application developer uses the Cataloged Exception in the exception handling section using the EEH API.

The first step for the application is to set up the Exception Catalog within the application by calling the API. The catalog must be loaded only once. The exceptions being generated by the application are validated against the catalog at runtime and a Cataloged Exception from the EEH internal catalog is thrown if the definitions do not match. This check avoids the creation of erroneous Exception Messages and to detect where the error might exist.

Periodically the Exception Catalog is updated and Exception Java class is generated to enable the developer to create the newly added exception. This process avoids errors in generating the Exception Message. Before the application is build for a specific version, the catalog is finalized for the version and imported to the database. The Exception Handling Daemon is started to consume Exception Messages from the failure queue and insert them in the database.

The eclipse plug-in may be installed to connect to the EEH database and analyze exceptions.

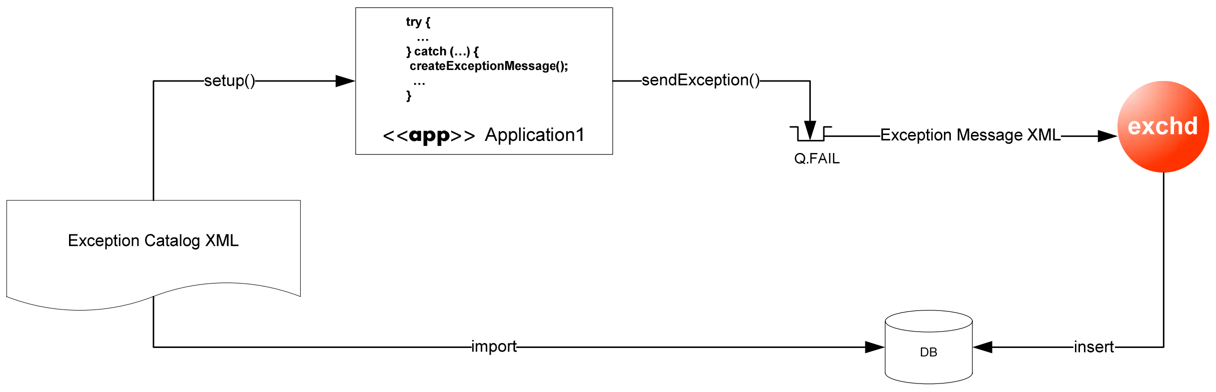


Figure 2 – EEH Implementation

The diagram illustrated the steps for using the EEH Implementation.

- Setup EEH_PATH
- Create EEH tables in the database
- Create Exception Catalog as the application is being developed
- Setup Exception Catalog within the application
- Generate Java class to facilitate coding of Cataloged Exceptions
- Application sends Exception Message to failure queue
- Exception Handling Daemon consumes the message and stores in the database
- Exception Analyzer plug-in can now analyze exceptions stored in the database

4. Design Pattern – Enterprise Exception Handling

The Enterprise Exception Handling is a design pattern that may be implemented in a number of ways. This paper further specifies the extensions of the pattern idea to a practical design and standardization. The intent of the paper is not just to explain the philosophy of the pattern but also to provide the Standardization required for the implementation of the pattern. This chapter is targeted to address the specifics of the design pattern. It has been broken down into the following sections:
Architecture – states the basic architecture for the pattern implementation
Message Specifications – states the different message specifications that flows through the ESB
Database Schema Specifications – states the database schema for persisting exceptions in the database
Features – states the salient features of the pattern implementation

4.1. Architecture

The architecture manifests itself in the following key concepts:

- ***Centralized Error Queue*** – all applications report exceptions to a common queue where an exception handling daemon processes the messages and sends them to a database
- ***Exception Daemon*** – the daemon process that sends the exception messages to the database
- ***Exception Database*** – central persistent storage for the exceptions
- ***Monitoring Agent*** – the common monitoring agent to report exceptions back to the central monitoring framework that is used throughout the organization
- **Expiry Handling** – exception messages should expire in case of unavoidable circumstances when the exceptions cannot be processed so that it does not fill up the common queue and cause other upstream problems
- **Exception Analyzer –** the Eclipse Editor plug-in for viewing exceptions and exception catalog entries, for creating new exceptions in the exception catalog and assigning them to applications, for re-queuing of exceptions, for manual editing of data before re-queuing of data and for statistical analysis of exceptions on the system
- **Re-Queuing** – exceptions with data and re-queue information may be re-queued at the point of failure
- **Exception Command Processor** – processes commands inside the exception handler daemon from the command queue

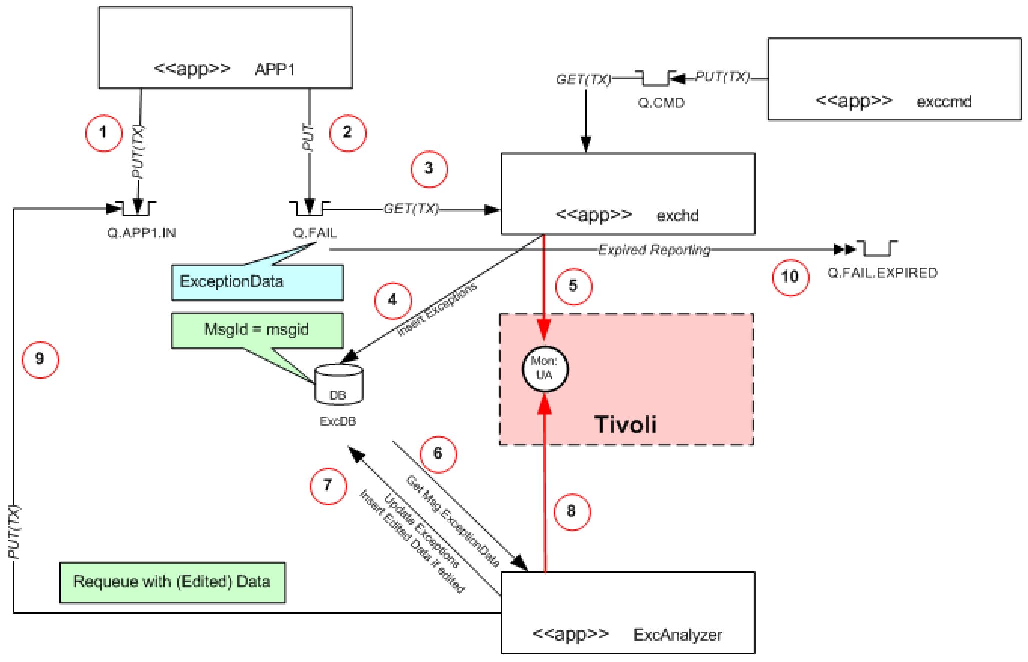


Figure 3 - Common Exception Handling Process

The diagram (Figure 3 - Common Exception Handling Process) shows how the exceptions will be handled throughout the organization from the common exception handling process. The basic idea is to have deep traceability of the exceptions occurring in the organization at any given point in time. Exception matrixes may be derived from such data in the future and corrective measures may be taken to minimize the occurrence of exceptions especially the critical ones. Also the introduction of new applications to the production environment may be checked for their integrity through the common exception handling mechanism. All exceptions will be cataloged based on the design. Exceptions would also feed into the monitoring system for altering based on certain situations. In the diagram Tivoli® monitoring system is shown as an example but the monitoring system selected by the organization should be integrated into the design. Since exceptions that are never acted upon for a long period of time looses its significance, the exceptions should expire with reporting option with full data to an achieved queue.

The example shows application APP1 failing to put to queue Q.APP1.IN and putting the exception in Q.FAIL. This is, however, true for all applications interacting with MQ or otherwise. The following are a detailed description of the steps:

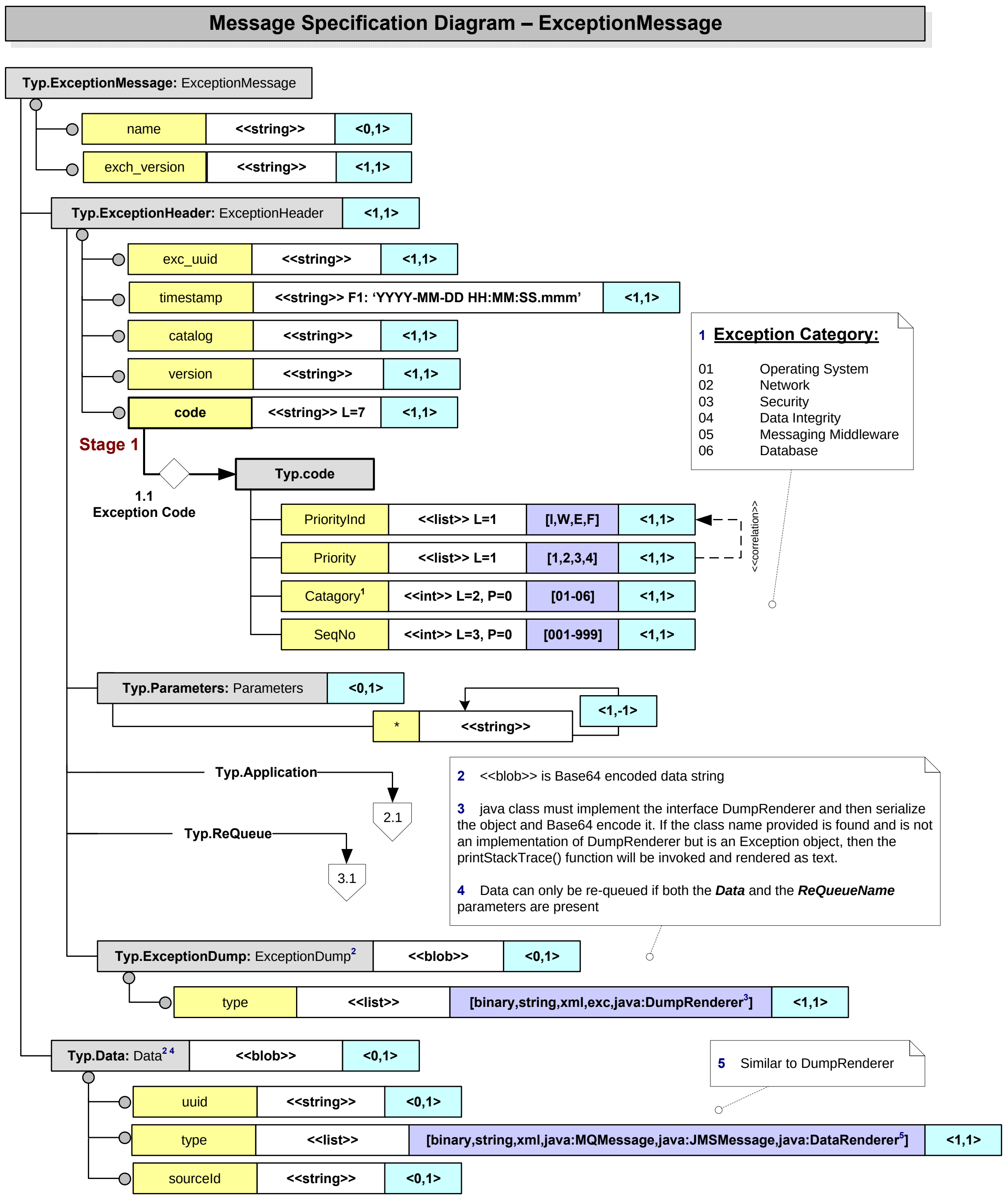
(1) APP1 tries to put a message in a transaction and fails
(2) APP1 puts Exception message into a queue outside the transaction with the following parameters:
  - **a.** **MQMD.Expiry = <time>**
  - **b.** **MQMD.Report = MQRO_EXPIRATION_WITH_FULL_DATA**
  - **c.** **MQMD.ReplyToQ = <queue>**

(3) The common Exception Handler application ExcHandler gets the Exception message transactionally
(4) Inserts the Exception information in the Exception database ExcDB
(5) ExcHandler notifies the Monitoring Agent (in case of Tivoli® Omegamon it should interface with the Universal Agent)
(6) The Exception Analyzer application ExcAnalyzer has the capability to analyze exceptions and update the data if necessary. In this step ExcAnalyzer gets the required information from the ExcDB
(7) ***(Optional step – only in the case of Data Integrity category of exceptions)*** ExcAnalyzer updates modified information in the ExcDB
(8) ExcAnalyzer notifies the Monitoring Agent to reset the exception
(9) ***(Optional step – only in the case of ReQueue parameter and Data parameter present)*** ExcAnalyzer puts the message in the original queue with the edited data if it had been modified or the unedited data
(10) Expired messages are forwarded to the archival queue Q.FAIL.EXPIRED by the queue manager

4.2. Message Specifications

Several message specifications have been standardized for implementation of the pattern and are described in brief below:
Exception Message – the specification that governs the format of exceptions sent throughout the organization
Exception Catalog – the specification that governs the format of the exception catalog that is loaded by the applications and also used for importing and exporting the catalog from the database from the exception utility
Command Message – the specification that governs the format in which commands are sent to the exception handler daemon and the command tool uses the standard for communicating with the daemon

The message specification diagram for the exception message is shown in the next diagram (Figure 4 – Message Specification for Exception Data).


Message Specification Diagram – ExceptionMessage
Typ.ExceptionMessage: ExceptionMessage
name <<string>> <0,1>
exch_version <<string>> <1,1>
Typ.ExceptionHeader: ExceptionHeader <1,1>
exc_uuid <<string>> <1,1>
timestamp <<string>> F1: 'YYYY-MM-DD HH:MM:SS.mmm' <1,1>
catalog <<string>> <1,1>
version <<string>> <1,1>
code <<string>> L=7 <1,1>
Stage 1
1.1
Exception Code
Typ.code
PriorityInd <<list>> L=1 [I,W,E,F] <1,1>
Priority <<list>> L=1 [1,2,3,4] <1,1>
Catagory1 <<int>> L=2, P=0 [01-06] <1,1>
SeqNo <<int>> L=3, P=0 [001-999] <1,1>
<<correlation>>
Typ.Parameters: Parameters <0,1>
* <<string>> <1,-1>
Typ.Application
2.1
Typ.ReQueue
3.1
Typ.ExceptionDump: ExceptionDump2 <<blob>> <0,1>
type <<list>> [binary,string,xml,exc,java:DumpRenderer3] <1,1>
Typ.Data: Data2 4 <<blob>> <0,1>
uuid <<string>> <0,1>
type <<list>> [binary,string,xml,java:MQMessage,java:JMSMessage,java:DataRenderer5] <1,1>
sourceId <<string>> <0,1>

1 **Exception Category:**

01	Operating System
02	Network
03	Security
04	Data Integrity
05	Messaging Middleware
06	Database

2 <<blob>> is Base64 encoded data string

3 java class must implement the interface DumpRenderer and then serialize the object and Base64 encode it. If the class name provided is found and is not an implementation of DumpRenderer but is an Exception object, then the printStackTrace() function will be invoked and rendered as text.

4 Data can only be re-queued if both the ***Data*** and the ***ReQueueName*** parameters are present

5 Similar to DumpRenderer

Figure 4 – Message Specification for Exception Data

Message Specification Diagram – ExceptionMessage (2.1)

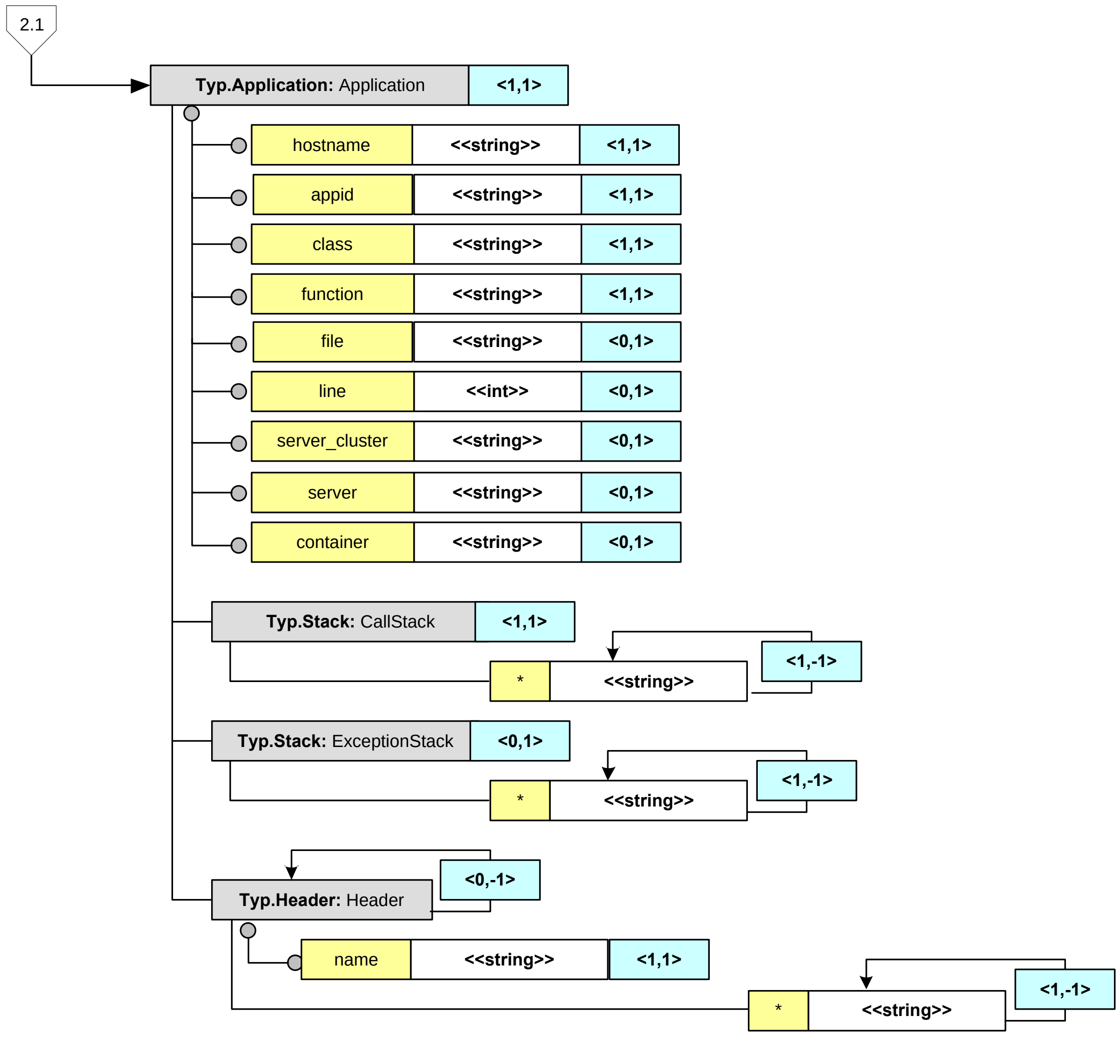


Figure 5 – Message Specification for Exception Data – Application section (2.1)

Message Specification Diagram – ExceptionMessage (3.1)

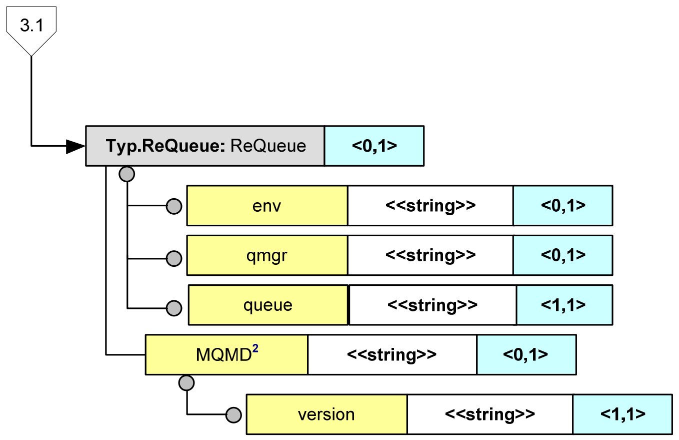


Figure 6 – Message Specification for Exception Data – Application section (3.1)

Details about the different fields in the Exception Message are listed in the table below. The data types are shown in the diagram.

Field		*Description*
exch_version		The version of the Exception Header. The Version for the current release is '***1.0.0***'.
name		Exception variable name for reference.
exc_uuid		UUID generated for the exception instance.
timestamp		The timestamp when the Exception was caught. The format of the timestamp is a string as **'YYYY-MM-DD HH:MM:SS'.**
catalog		Message Catalog is the common repository of all error messages for the subsystem. This name should always be '***ExcCat.HSBC***' for the each project. This assures the existence of other projects without Exception Code clashes between the different subsystems
version		Version of the Exception Catalog.
code		Unique exception code for the project. Exception Codes are not meant to replace the Product Exception Codes but to group them together under several categories for common error handling implemented in the subsystem.
	PriorityInd	The Priority Indicator is a one character field corresponding to the Priority. It is one of the values '***I***', '***W***', '***E***', or '***F***'.
	Priority	This the PriorityChar field represented in integral values for applications that handle Exception Codes as integers, i.e. 1 ε Information 2 ε Warning 3 ε Error 4 ε Fatal
	Category	The exception codes are categorized in the following groups – Operating System ('01') e.g. memory allocation, file system, kernel objects, etc. Network ('02') e.g. routing exceptions, network traffic, etc. Security ('03') e.g. service access control, connectivity, etc. Data Integrity ('04') e.g. Common header format error, etc. Messaging Middleware ('05') e.g. MQ Connection, MQ Object Access, etc. Database ('06') e.g. DB Connection, Insert failure, etc.
	SeqNo	This is a serial number for the Exception in the class and category.
Parameters		`Optional`. The parameters are defined as name-value pairs as children of Parameters. The number of parameters passed depends on the particular Exception Code in the Message Catalog. E.g., for Database Connection problems the Parameter[] would contain 3 parameters – SQLSTATE, SQLCODE, SQLERRORTEXT. It is represented as a name-value pair and is fixed for the exception code.

Application		Section for application data
	hostname	The host machine name or the static IP address of the machine where the Exception occurred.
	appid	The application identifier of the program that generated the Exception condition and sent out the report. In case of the message broker this field will contain the name of the Message Flow's application id.
	class	The class that generated the Exception
	function	The function name where the Exception was generated. In the case of the message broker it is the name of the node that detected the Exception.
	file	File name where the exception was detected.
	line	Line number where the exception was detected.
	server_cluster	Cluster name the server hosting the application belongs to. E.g. Cell for WebSphere® Application Server, etc.
	server	Server name where the application is hosted. E.g. J2EE Application Server name, WebSphere® Message Broker name
	container	Container name inside the server hosting the application. E.g. Container name for the J2EE Application server, Execution Group for the Message Broker, etc.
	Header	`Optional.` Application Header Section may be added for any application related information. that may be required to be passed as meaningful information for the exception condition. The header parameters are defined as name-value pairs as children of Header.
	name	Name of the application header
	CallStack	The Call Stack is automatically generated from the Cataloged Exception constructor. It is the stack from where the exception is being generated.
	ExceptionStack	`Optional.` The Exception Stack is the Call Stack of the exception object encapsulated in the Cataloged Exception.
ReQueue		`Optional.` The ReQueue Section is an optional section that provides information for reintroducing the data in the originating queue and is populated by the process throwing the exception. If necessary, it indicates to the error handling process the queue to which the message should be re-queued once the exception is corrected.
	env	The environment where the exception occurred. This parameter is associated with the application and the data can re-queued in the different environment for debugging or tracing while analyzing the problem. Default environment is a zero length string.
	qmgr	`Optional.` Optional name of the queue manager to be re-queued on
	queue	Name of the queue where the data may be re-queued.
	MQMD	`Optional.` Message Descriptor of the original message for re-queuing. The MQMD is converted into an element and each of the property to an attribute in the XML. This conversion makes the values of the MQMD easily readable.
ExceptionDump		This is an optional field. This field reports any system dump or nested exceptions thrown from the broker for determining the exception that occurred. This is a fixed length field if used and the length of the field should be expressed as a 10 digit integer string. For binary type or serialized java objects, the data is Base64 encoded.
	type	The type of exception dump may be of the following flavors: ***binary*** – binary data dump ***string*** – any string type that may provide exception information ***xml*** – exception dumps represented in XML form excluding the XML declaration ***exc*** – incorporates an exception header. This type of dump is created by the exchd in case of exceptions trapped in the exchd while processing an exception. The type is not limited to the exchd. ***java:<Class>*** – serialized and Base64 encoded java class implementation of the interface DumpRenderer or an Exception class object.

Data		`Optional.` The data that may be re-queued at the point of exception. Typically this is the incoming data for the transaction so that the transaction can be re-run after fixing the problem. The data may also be added purely for reference.
	uuid	`Optional.` Universally Unique Identifier to identify the data in the system. This may be used for analyzing if the same data is encountering multiple or repeated exceptions. Automatic re-queuing of data may also use it for infinite loop control within its rules. If the uuid is provided along with the sourceId, the data may not be provided and passed along. This reduces the size of the exception message.
	sourceId	`Optional.` The source of the data already stored in a table. The source id may be configured for the application similar to configuring the queue manager. **`This feature will be enabled in a future release.`**
	type	The type of exception dump may be of the following flavors: ***binary*** – binary data dump ***string*** – any string type that may provide exception information ***xml*** – exception dumps represented in XML form excluding the XML declaration ***java:<Class>*** – serialized and Base64 encoded java class implementation of the interface DataRenderer. ***java:MQMessage*** or ***java:JMSMessage*** – serialized and Base64 encoded for native MQ or JMS support.

An example of an exception message:

```
<?xml version="1.0" encoding="UTF-8"?>
<exc:ExceptionMessage exch_version="1.0.0" name="WMQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:com.ibm.mq.exc ExceptionMessage.xsd" xmlns:exc="urn:com.ibm.mq.exc">
        <ExceptionHeader version="1.0.0" catalog="ExcCat.MQTT" code="F405001"
                        timestamp="2009-05-11 18:29:35.787 GMT"
                        exc_uuid="d6a4efcf-3e69-4d26-8aaf-b357be910fe6">
                <Parameters>
                        <Reason> 2085 </Reason>
                </Parameters>
                <Application hostname="Ronz-Master-2009.ron.ibm.com" appid="MQTT"
                          class="com.ibm.mq.test.ds.QueueSource" function="handleMQConn"
                          file="QueueSource.java" line="170" >
                </Application>
                <ExceptionDump type="java:com.ibm.mq.MQException">

                rO0ABXNyABZjb20uaWJtLm1xLk1RRXhjZXB0aW9uZdb+evXEtIkCAAhJAA5jb21wbGV0aW9uQ29kZUkA
BW1zZ0lkSQAKbnVtSW5zZXJ0c0kACnJlYXNvbkNvZGVMAAdpbnNlcnQxdAASTGphdmEvbGFuZy9TdHJpbmc7TAAHaW5zZ
XJ0MnEAfgABTAALb3N0ck1lc3NhZ2VxAH4AAUwAB3Y3bXNnSWRxAH4AAXhyABNqYXZhLmxhbmcuRXhjZXB0aW9u0P0fPh
o7HMQCAAB4cgATamF2YS5sYW5nLlRocm93YWJsZdXGNSc5d7jLAwADTAAFY2F1c2V0ABVMamF2YS9sYW5nL1Rocm93
YWJsZTtMAA1kZXRhaWxNZXNzYWdlcQB+AAFbAApzdGFja1RyYWNldAAeW0xqYXZhL2xhbmcvU3RhY2tUcmFjZUVsZW1lb
nQ7eHBwcHVyAB5bTGphdmEubGFuZy5TdGFja1RyYWNlRWxlbWVudDsCRio8PP0iOQIAAHhwAAAACXNyABtqYXZhLmxhb
mcuU3RhY2tUcmFjZUVsZW1lbnRhCcWaJjbdhQIABEkACmxpbmVOdW1iZXJMAA5kZWNsYXJpbmdDbGFzc3EAfgABTAAIZm
lsZU5hbWVxAH4AAUwACm1ldGhvZE5hbWVxAH4AAXhwAAABPHQAGGNvbS5pYm0ubXEuTVFFEZXN0aW5hdGlvbnQAEk1R
RGVzdGluYXRpb24uamF2YXQABG9wZW5zcQB+AAkAAAEEdAASY29tLmlibS5tcS5NUVF1ZXVldAAMTVFRdWV1ZS5qYXZh
dAAGPGluaXQ+c3EAfgAJAAAKrnQAGWNvbS5pYm0ubXEuTVFRdWV1ZU1hbmFnZXJ0ABNNUVF1ZXVlTWFuYWdlci5qYXZ
hdAALYWNjZXNzUXVldWVzcQB+AAkAAArKdAAZY29tLmlibS5tcS5NUVF1ZXVlTWFuYWdlcnQAE01RUXVldWVNYW5hZ2Vy
LmphdmF0AAthY2Nlc3NRdWV1ZXNxAH4ACQAAAJB0AB5jb20uaWJtLm1xLnRlc3QuZHMuUXVldWVTb3VyY2V0ABBRdWV1
ZVNvdXJjZS5qYXZhdAAMaGFuZGxlTVFDb25uc3EAfgAJAAAAKHQAHmNvbS5pYm0ubXEudGVzdC5kcy5RdWV1ZVNvdXJjZ
XQAEFF1ZXVlU291cmNlLmphdmF0AAhzZW5kRGF0YXNxAH4ACQAAAFR0ABljb20uaWJtLm1xLnRlc3QuUmVndWxhdG9yd
AAOUmVndWxhdG9yLmphdmF0AANydW5zcQB+AAkAAAIPdAAVamF2YS51dGlsLlRpbWVyVGhyZWFkdAAKVGltZXIuamF2Y
XQACG1haW5Mb29wc3EAfgAJAAAAB3XQAFWphdmEudXRpbC5UaW1lclRocmVhZHQAClRpbWVyLmphdmF0AANydW54AA
AAAgAAAAAAAAAAAAAAIJXBwdAAsTVFKRTAwMTogQ29tcGxldGlvbiBDb2RlICcyJywgUmVhc29uICcyMDg1Jy50AAA=

                </ExceptionDump>
        </ExceptionHeader>
</exc:ExceptionMessage>
```

Message Specification Diagram – ExceptionCatalog

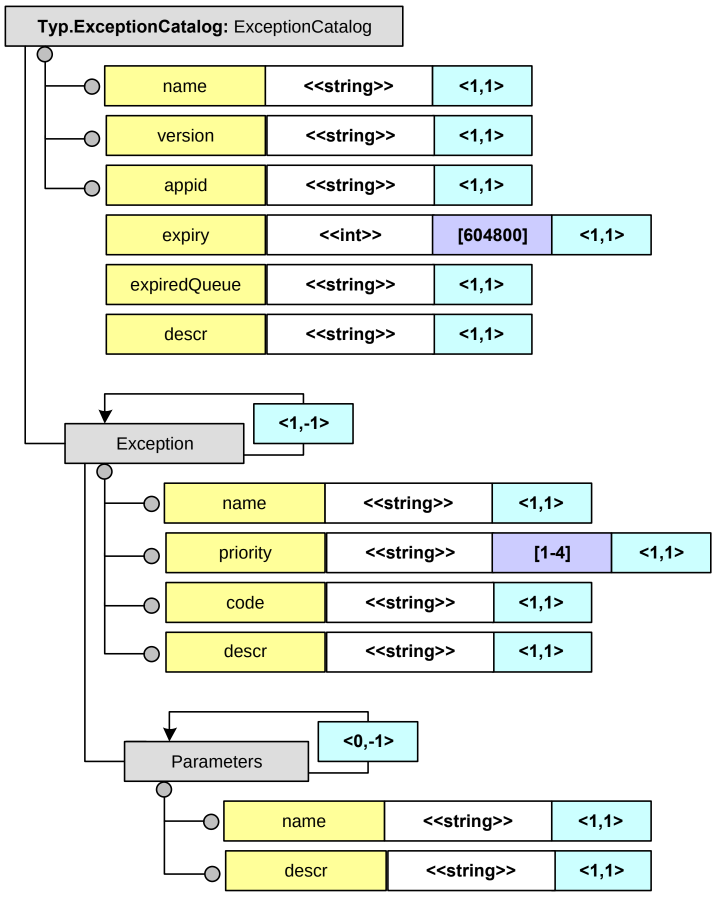


Figure 7 – Message Specification for Exception Catalog

Details about the different fields in the Exception Catalog are listed in the table below. The data types are shown in the diagram.

Field		*Description*
name		The Exception Catalog name.
version		The Exception Catalog version.
appid		The application identifier that uses the exception catalog.
expiry		Expiry of the exception message on the queue in seconds (default **604800 =** 7 days)
expiredQueue		The expired queue name where the message will be forwarded to in case the exception message expired after the expiry specified.
descr		Description of the Exception Catalog
Exception		Section for the exception information
	name	Name of the exception. This disassociates the Application program from the exception code
	priority	The default priority of the exception. This is used to generate the Exception through the generated class without having to mention the priority. The priority of an exception can be changed at any time based on situations for exception promotion or demotion.
	code	The exception code associated with the exception.
	descr	Description for the exception providing detailed information regarding the exception.
	Parameters	`Optional.` The parameters are defined as name-value pairs as children of Parameters. The number of parameters passed depends on the particular Exception Code in the Message Catalog. E.g., for Database Connection problems the Parameter[] would contain 3 parameters – SQLSTATE, SQLCODE, SQLERRORTEXT. It is represented as a name-value pair and is fixed for the exception code.

Message Specification Diagram – CommandMessage

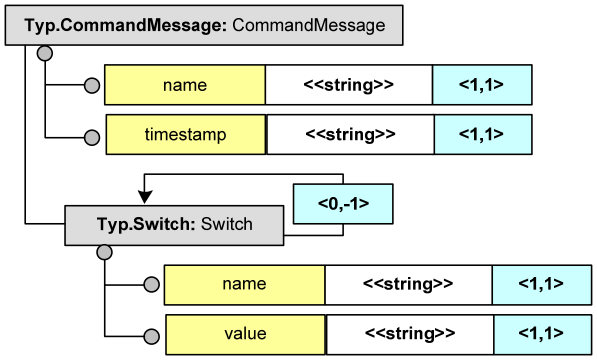


Figure 8 –Message Specification for Command Message

Details about the different fields in the Command Message are listed in the table below. The data types are shown in the diagram.

Field	*Description*
name	The name of the command. E.g., shutdown for the Shutdown command sent to the exception handling daemon.
timestamp	The timestamp when the Command was created. The format of the timestamp is a string as **'YYYY-MM-DD HH:MM:SS'**.
Switch	`Optional.` Switches are parameters sent for the particular command. E.g., the command may be operated under multiple modes, viz. immediate or quiesce.
name	Name of the switch.
value	Value of the switch.

4.3. Database schema Specifications

The following schema diagram illustrates how the Exception database is designed and all the relationships are shown.

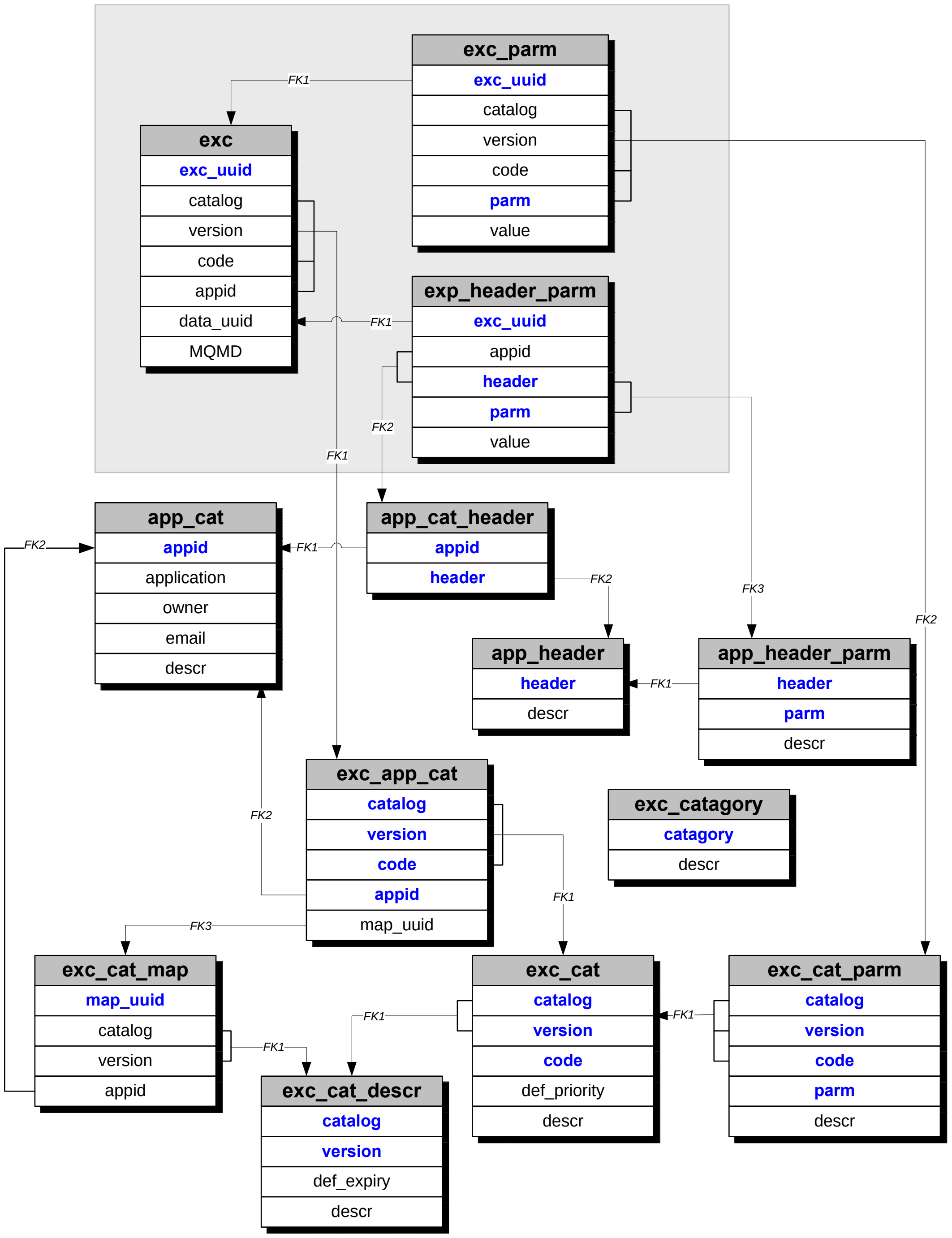


Figure 9 - Exception Database Schema

The three tables on the top right corner marked with a grey rectangle indicate the tables that hold runtime exception information. All the exceptions that are generated at runtime are being sent to the exception handling daemon and are inserted into the **exc**, **exc_parm** and **exc_header_parm** tables. The rest of the tables hold static data. The **exc_cat** and the **exc_cat_parm** tables hold the exception catalog information. The **app_cat** table holds the application catalog. The application headers are stored in the **app_header** and **app_header_parm** tables. Each application header may be associated with any of the applications defined in the system. The relationship is maintained in the **app_cat_header** table. Each application is also associated with any of the exceptions in the exception catalog. The relationship is maintained in the **exc_app_cat** table.

The following are the description of all the configuration tables for the Exception Catalog Database.

exc_cat_descr – Exception Catalog Information		
Column	*Type*	*Description*
catalog	**Varchar(30)**	Exception Catalog name for the project or organization
version	**Varchar(10)**	Exception Catalog version
def_expiry	**Integer**	Default expiry, in sec., of the Exception messages in the Catalog
descr	**Varchar(512)**	Description of the Exception Catalog
Key	*Reference*	*Columns*
PK		catalog, version

exc_cat – Exception Catalog Listing		
Column	*Type*	*Description*
catalog	**Varchar(30)**	Exception Catalog name for the project or organization
version	**Varchar(10)**	Exception Catalog version
code	**Char(5)**	Exception code: First 2 char for the category and the rest 3 are serial number
name	**Varchar(30)**	Exception name - used as Exception Variable for code isolation
def_priority	**Integer**	Default priority of the Exception
descr	**Varchar(512)**	Description of the Exception
Key	*Reference*	*Columns*
PK		catalog, version, code
UK		catalog, version, name
FK1	**EXC_CAT_DESCR**	catalog, version

exc_cat_parm – Exception Catalog Parameters		
Column	*Type*	*Description*
catalog	**Varchar(30)**	Exception Catalog name for the project or organization
version	**Varchar(10)**	Exception Catalog version
code	**Char(5)**	Exception code: First 2 char for the category and the rest 3 are serial number
parm	**Varchar(30)**	Parameter for the Cataloged Exception
descr	**Varchar(512)**	Description of the Exception parameter
Key	*Reference*	*Columns*
PK		catalog, version, code, parm
FK1	**EXC_CAT**	catalog, version, code

exc_catagory – Exception Category		
Column	*Type*	*Description*
catagory	**Char(2)**	Excecption Category Supported. 01 - System, 02- Security, 03 - Network, 04 - Data Integrity, 05 - Middleware, 06 - Database, more may be added
descr	**Varchar(512)**	Description of the Exception Category
Key	*Reference*	*Columns*
PK		catagory

app_cat – Application Catalog		
Column	*Type*	*Description*
appid	**Varchar(10)**	Application Identifier
application	**Varchar(50)**	Application name
owner	**Varchar(100)**	Name of the Application Owner
email	**Varchar(30)**	E-mail of the Application Owner
descr	**Varchar(512)**	Description of the Application
Key	*Reference*	*Columns*
PK		appid

app_header – Application Catalog Header		
Column	*Type*	*Description*
header	**Varchar(30)**	Application Header
descr	**Varchar(512)**	Description of the Applicaion Header
Key	*Reference*	*Columns*
PK		header

app_header_parm – Application Catalog Header Parameter		
Column	*Type*	*Description*
header	**Varchar(30)**	Application Header
parm	**Varchar(30)**	Application Header Parameter
descr	**Varchar(512)**	Description of the Application Header Parameter
Key	*Reference*	*Columns*
PK		header, parm
FK1	**APP_HEADER**	header

app_cat_header – Association of the Application with the Application Header		
Column	*Type*	*Description*
appid	**Varchar(30)**	Application Identifier
header	**Varchar(30)**	Application Header
Key	*Reference*	*Columns*
PK		appid, header
FK1	**APP_CAT**	appid
FK2	**APP_HEADER**	header

exc_cat_map – Association of the Exception Catalog for the Application		
Column	***Type***	***Description***
map_uuid	**Char(36)**	Universally Unique Identifier for the association between Exception Catalog and the Application
catalog	**Varchar(30)**	Exception Catalog name for the project or organization
version	**Varchar(10)**	Exception Catalog version
appid	**Varchar(30)**	Application Identifier
Key	***Reference***	***Columns***
PK		map_uuid
FK1	**EXC_CAT_DESCR**	catalog, version
FK2	**APP_CAT**	appid

exc_app_cat – Association of the Exception for the Application		
Column	***Type***	***Description***
catalog	**Varchar(30)**	Exception Catalog name for the project or organization
version	**Varchar(10)**	Exception Catalog version
code	**Char(5)**	Exception code: First 2 char for the category and the rest 3 are serial number
appid	**Varchar(30)**	Application Identifier
map_uuid	**Char(36)**	Universally Unique Identifier for the association between Exception Catalog and the Application
Key	***Reference***	***Columns***
PK		catalog, version, code, appid
FK1	**EXC_CAT**	catalog, version, code
FK2	**APP_CAT**	appid
FK3	**EXC_CAT_MAP**	map_uuid

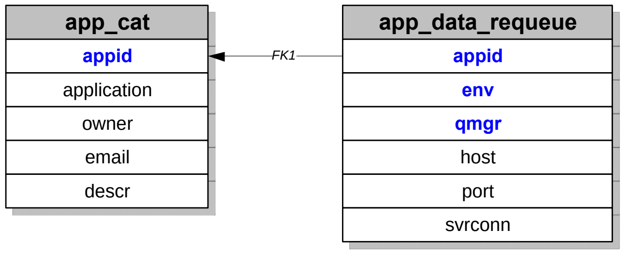


Figure 10 - Application Data Re-Queue

app_data_requeue – ReQueuing information for connecting to the MQ QMgr		
Column	***Type***	***Description***
appid	**Varchar(10)**	Application Identifier
env	**Varchar(25)**	Environment whare the Queue Manager is defined
qmgr	**Varchar(32)**	Queue Manager name
hostname	**Varchar(512)**	Hostname or IP address where Exception was generated
port	**Integer**	Port for connecting to the Queue Manager listener
svrconn	**Varchar(20)**	Name of the SVRCONN channel for client connections
Key	***Reference***	***Columns***
PK		appid,env,qmgr
FK1	**APP_CAT**	appid

The following are the exception instance tables for capturing the real-time exceptions generated in the system.

exc – Exception instance generated

Column	Type	Description
exc_uuid	Char(36)	Universally Unique Identifier for the Exception instance
timestamp	DateTime	Timestamp when the exception was generated
catalog	Varchar(30)	Exception Catalog name for the project or organization
version	Varchar(10)	Exception Catalog version
code	Char(5)	Exception code: First 2 char for the category and the rest 3 are serial number
priority	Integer	Exception priority: 1 - Information, 2 - Warning, 3 - Error, 4 - Fatal
hostname	Varchar(512)	Hostname or IP address where Exception was generated
appid	Varchar(30)	Application Identifier
class	Varchar(512)	Name of the class that encountered the exception
function	Varchar(512)	Function name where Exception was generated
file	Varchar(512)	Program file name of Exception source
line	Integer	Line number of the program where Exception was generated
server_cluster	Varchar(512)	Cluster name for the server cluster. NULL for non-clustered servers, e.g. Cell for WebSphere Application Server
server	Varchar(512)	Server name for the server hosting the application. NULL for stand-alone applications, e.g. J2EE Application Server, WebSphere® Message Broker
container	Varchar(512)	Server container for hosting the application. NULL for stand-alone applications or non-container based applications, e.g. EJB container for J2EE Application Server, Execution Group for WebSphere® Message Broker
MQMD	Varchar(1536)	MQ Message Descriptor for the original message encoded as XML attribute
MQMD_ver	Integer	Version of the MQ Message Descriptor for the original message
env	Varchar(25)	Environment whare the Queue Manager is defined
qmgr	Varchar(32)	Queue Manager name for re-queuing of the input message
queue	Varchar(32)	Queue name for re-queuing of the input message
requeue_status	Timestamp	Timestamp when the data was last re-queued to the Source Queue. Null if never re-queued
call_stack	Blob	Call Stack from the Application when the Exception was thrown
exc_stack	Blob	Exception stack of the original exception
dump_type	Varchar(512)	The type of exception dump may be of the following flavors: ***binary*** – binary data dump ***string*** – any string type that may provide exception information ***xml*** – exception dumps represented in XML form excluding the XML declaration ***exc*** – embedded exception ***java:<Class>*** – serialized and Base64 encoded java class implementation of the interface DumpRenderer or an Exception class object.
dump	blob	Exception Dump

data_uuid	**Varchar(32)**	Universally Unique Identifier for the Data in the message. If the data is already defined in a table with this uuid, define a foreign key to the table
data_sourceId	**Varchar(512)**	Source Identification and access plan for the Source Data
data_type	**Varchar(512)**	The type of exception dump may be of the following flavors: ***binary*** – binary data dump ***string*** – any string type that may provide exception information ***xml*** – exception dumps represented in XML form excluding the XML declaration ***java:<Class>*** – serialized and Base64 encoded java class implementation of the interface DataRenderer. ***java:MQMessage*** or ***java:JMSMessage*** – serialized and Base64 encoded for native MQ or JMS support.
data	**blob**	Source Data
mod_data	**blob**	Modified Data
mod_usr	**Varchar(256)**	Modifier User Identifier
mod_time	**DateTime**	Modification timestamp
Key	***Reference***	***Columns***
PK		exc_uuid
FK1	**EXC_APP_CAT**	catalog, version, code, appid

exc_parm – Exception instance parameters

Column	***Type***	***Description***
exc_uuid	**Char(36)**	Universally Unique Identifier for the Exception instance
catalog	**Varchar(30)**	Exception Catalog name for the project or organization
version	**Varchar(10)**	Exception Catalog version
code	**Char(5)**	Exception code: First 2 char for the category and the rest 3 are serial number
parm	**Varchar(30)**	Parameter for the Exception instance
value	**Varchar(32)**	Value for the exception parameter
Key	***Reference***	***Columns***
PK		exc_uuid, parm
FK1	**EXC**	exc_uuid
FK2	**EXC_CAT_PARM**	Catalog, version, code, parm

exc_header_parm – Application Headers for the Exception instance generated

Column	***Type***	***Description***
exc_uuid	**Char(36)**	Universally Unique Identifier for the Exception instance
appid	**Varchar(10)**	Application Identifier
header	**Varchar(30)**	Application Header
parm	**Varchar(30)**	Application Header Parameter
value	**Varchar(32)**	Value for the application header parameter
Key	***Reference***	***Columns***
PK		exc_uuid, parm
FK1	**EXC**	exc_uuid
FK2	**EXC_CAT_PARM**	Catalog, version, code, parm

exc_data_mod_hist – Data Modification History for the generated Exception		
Column	*Type*	*Description*
exc_uuid	**Char(36)**	Universally Unique Identifier for the Exception instance
mod_time	**Timestamp**	Modification timestamp
mod_app	**Varchar(256)**	Application that is modifying the data. In case of Data Handler within the exchd for automatic fixes to the data the registered plugin class will be stored
mod_usr	**Varchar(256)**	Modifier User Identifier
mod_type	**Varchar(30)**	Modification type: data, requeue
mod_data	**Blob**	Modified Data
Key	*Reference*	*Columns*
PK		exc_uuid,mod_time
FK1	**EXC**	exc_uuid

exc_data_req_hist – Data Re-Queuing History for the generated Exception		
Column	*Type*	*Description*
exc_uuid	**Char(36)**	Universally Unique Identifier for the Exception instance
mod_time	**Timestamp**	Modification timestamp
appid	**Varchar(10)**	Application Identifier
env	**Varchar(25)**	Environment where the Queue Manager is defined
qmgr	**Varchar(32)**	Queue Manager name for re-queuing of the input message
Key	*Reference*	*Columns*
PK		exc_uuid,mod_time
FK1	**EXC_DATA_MOD_HIST**	exc_uuid,mod_time

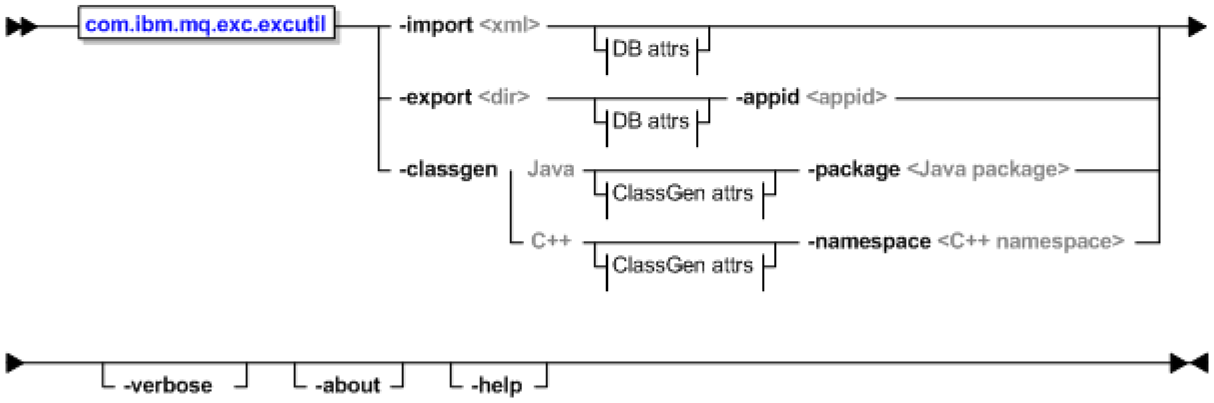
5. Exception Handler Runtime and Utilities:

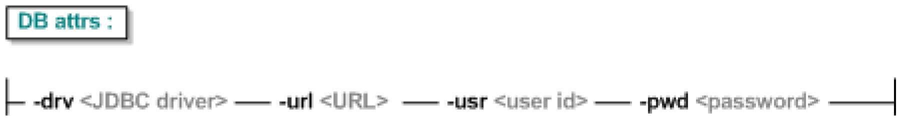
5.1. Exception Handler Utility

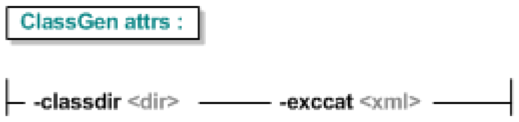
A utility program has been provided with the exception handling package for facilitating three major functions:

- Importing exception catalog for the application into the database
- Exporting exception catalogs in the database for individual applications
- Generating the exception catalog class for facilitating exception generation for the applications

The syntax of the exception handling utility is as follows:







```
com.ibm.mq.exc.excutil
            [((-import <xml>| -export <dir>) -drv <JDBC driver> -url <URL>
            -usr <user id> -pwd <password> [-appid <appid>])|
            (-classgen ('Java'|'C++') -classdir <dir> -exccat <xml>
            [(-package <Java package> | -namespace <C++ namespace>)])]
            [-verbose] [-about] [-help]
```

-import
The XML file in to be imported conforming to the schema ExceptionCatalog.xsd.

-export
The directory where the exported Exception Catalog(s) will be saved.
The file name created follows the convention:
[<AppId>_]<Catalog>_<Catalog Version>.xml
If no AppId is specified in the export command, all the Exceptions for the Catalog are listed and the file name does not mention an AppId.
The exception catalog file(s) saved conform to the schema ExceptionCatalog.xsd.

-drv
JDBC driver, e.g. com.ibm.db2.jcc.DB2Driver.

-url
JDBC URL pointing to the exception database,
e.g. jdbc:db2://10.20.20.10:50000/EXCCAT.

-usr
User Id for the exception database.

-pwd
Password for the exception database.

-appid
Appication Identifier for the application whose related exceptions are to be exported. This is an optional parameter only relevent for the export option. If omitted the complete exception catalog is exported.

-classgen
Generate class file in the Language specified.

-classdir
The directory where the class file will be generated.

-exccat
The exception catalog XML from which the classfile is to be generated.
This parameter is only used when the -classgen parameter is used without either the -import or -export parameters.

-verbose
Optional parameter only to be used for debugging if errors are not explicit.

-about
Version and related information.

[1]Examples[2]:

- Importing the exception catalog into the database:

```
call bin/win/excutil.bat -import samp/xml/ExcCat.MQST.xml -drv
"com.ibm.db2.jcc.DB2Driver" -url "jdbc:db2://localhost:50000/EXCCAT" -usr arunava
-pwd ********
```

- Exporting the exception catalog from the database:

```
call bin/win/excutil.bat -export samp/xml –appid MQST -drv "com.ibm.db2.jcc.DB2Driver"
-url "jdbc:db2://localhost:50000/EXCCAT" -usr arunava -pwd ********
```

- Generating a Java class from the exception catalog XML:

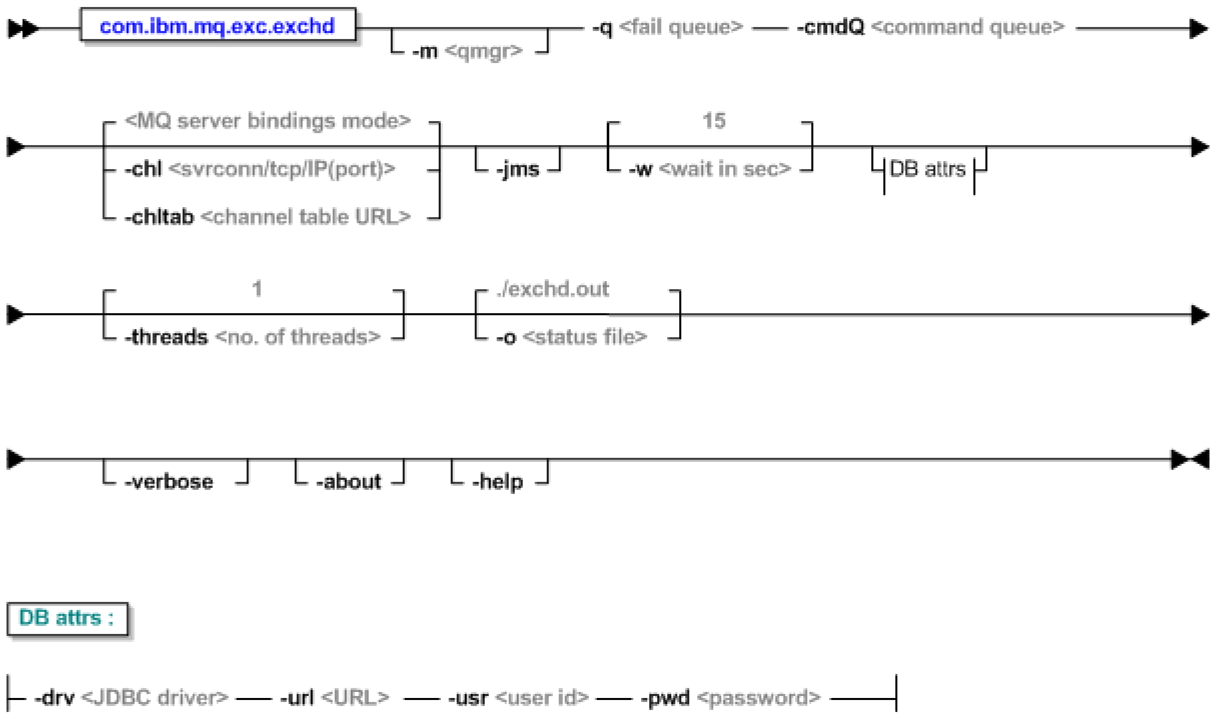
```
call bin/win/excutil.bat -classgen Java -classdir Tool/ETR/_DEV/MQTT/com/ibm/mq/test
-exccat Tool/ETR/_DEV/MQTT/exccat/ExcCat.MQTT.xml -package com.ibm.mq.test
```

[1] **Remember to set up the EEH_PATH environment variable to the EEH installation path.**

[2] **Use the excutil.bat or excutil.sh scripts in the bin/win or bin/unix directories to facilitate setting up of the jar files for EEH. Product jar files, viz. WMQ, DB2, etc. must be setup for the implementation.**

5.2. Exception Handler Daemon

The Exception Handler Daemon utility runs as a independent multi-threaded Java process to parse and store exception information from the queue to the database. The daemon may be invoked from inside any Java program as well if so desired. The number of threads may be increased or decreased at the start of the application. The wait on the queue also determines how quickly the daemon may be shutdown. The maximum time to shutdown the daemon is the wait period on the queue. The daemon can be horizontally or vertically scaled if required.



com.ibm.mq.exc.exchd
-drv <JDBC driver> -url <URL>
-usr <user id> -pwd <password>
-q <queue> [-m <qmgr>] [-w <wait in sec>] [-jms]
[(-chl <svrconn/tcp/IP(port)> | -chltab <channel table URL>)]
-cmdQ <command queue> [-threads <no. of threads>]
[-o <status file>]
[-verbose] [-about] [-help]

-drv
JDBC driver, e.g. com.ibm.db2.jcc.DB2Driver.

-url
JDBC URL pointing to the exception database,
e.g. jdbc:db2://10.20.20.10:50000/EXCCAT.

-usr
User Id for the exception database.

-pwd
Password for the exception database.

-q
Name of the Exception Queue.

-m
Optional. Name of the Exception Queue Manager.

-w
Optional. Wait in seconds on the exception queue before it is checked again. This also determines the maximum time required to shutdown the daemon gracefully by running the shutdown command.
[Default value is 15]

-jms
Optional. JMS messaging is turned on.

-chl
Optional. Client channel definition to connect to the Queue Manager.
The format is the same as the MQSERVER environment variable:
svrconn/tcp/IP(port)
If both -chl and -chltab are skipped MQ Binding mode is used.

-chltab
Optional. URL to the client channel table.
If both -chl and -chltab are skipped MQ Binding mode is used.

-cmdQ
The Command Queue for the daemon to process Command messages.

-threads
Optional. Number of threads for processing exceptions.
[Default value is 1].

-o
Optional. Output status file for the daemon's log.

-verbose
Optional parameter only to be used for debugging if errors are notexplecit.

-about
Version and related information.

[3]Examples[4]:

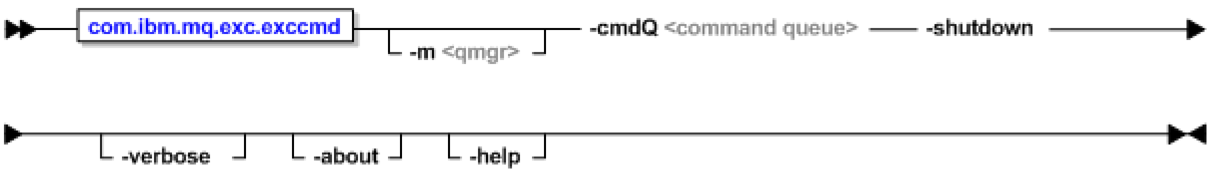
- Running the exception handling daemon:

```
call bin/win/exchd.bat -drv "com.ibm.db2.jcc.DB2Driver"
-url "jdbc:db2://localhost:50000/EXCCAT" -usr arunava -pwd ****** -q Q.FAIL -m QM.HUB
-cmdQ Q.CMD
```

[3] **Remember to set up the EEH_PATH environment variable to the EEH installation path.**
[4] **Use the shutdown.bat or sutdown.sh scripts in the bin/win or bin/unix directories to facilitate setting up of the jar files for EEH. Product jar files, viz. WMQ, DB2, etc. must be setup for the implementation.**

5.3. Exception Handler Command

The Exception Handler Command utility is designed to send commands to the Exception Handler Daemon at runtime. The utility converts the command line parameters to the Exception Command Message specification and sends the message to the Command Queue. The only command implemented in this release is **shutdown**. This command is sent to the daemon to quiesce MQ and Database transactions, disconnect and shut itself down. This is the recommended way of stopping the Exception Handler Daemon.



com.ibm.mq.exc.exccmd
[-m <qmgr>] -cmdQ <command queue>
-shutdown
[-verbose] [-about] [-help]

-m	Optional parameter for the name of the Queue Manager. Not required if a default queue manager is defined for the node.
-cmdQ	Name of the Command Queue that the Exception Handling Daemon is listening on.
-shutdown	Only command supported in this release is shutdown
-verbose	Optional parameter only to be used for debugging if errors are notexplecit.
-about	Version and related information.

[5]Examples[6]:

- Running the shutdown command to bring down the exception handling daemon:

```
call bin/win/shutdown.bat -m QM.HUB -cmdQ Q.CMD -shutdown
```

[5] **Remember to set up the EEH_PATH environment variable to the EEH installation path.**
[6] **Use the shutdown.bat or sutdown.sh scripts in the bin/win or bin/unix directories to facilitate setting up of the jar files for EEH. Product jar files, viz. WMQ, DB2, etc. must be setup for the implementation.**

6. Setting up the Exception Catalog:

Every application defines its exception catalog against which all the exceptions are reported in the system. The catalog must be defined in the exception database. However, the application is not dependent on the database existence. This isolates the application from the database dependency. The exception XML may be loaded into the database or dumped from the database using the exception utility (**excutil**). The exception catalog XML for the application is limited to the exceptions that the particular application may generate and the application must be set up for all these exceptions in the database.

Below is the sample[7] exception catalog listing for the application defined with the id **MQST**. The Exception catalog name is **ExcCat.MQST** and version **1.0.0**.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--

 Licensed Materials - Property of IBM
 (C) Copyright IBM Corp. 2009 All Rights Reserved.
 US Government Users Restricted Rights - Use, duplication or
 disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

 Created January 2008, 2012

  -->
<exc:ExceptionCatalog name="ExcCat.MQST" version="1.0.0" appid="MQST"
     expiry="604800" expiredQueue="Q.EEH.EXPIRED"
     descr="Exception Catalog for MQST Application"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:com.ibm.mq.exc ../schema/ExceptionCatalog.xsd"
      xmlns:exc="urn:com.ibm.mq.exc">

<Exception name="EXC_RULES_SYNTAX" priority="3" code="4001" descr="Rules Syntax Error!">
  <Parameter name="Element" descr="Element at which the syntax is incorrect." />
  <Parameter name="Value" descr="Value of the element containing incorrect syntax." />
  <Parameter name="Position" descr="Position where the syntax violation was detected in the
                                    element value." />
  <Parameter name="Reason" descr="Reason for the syantax violation to occur." />
</Exception>
<Exception name="EXC_XML_VALIDATION" priority="3" code="4002" descr="XML Validation Error!">
  <Parameter name="PublicId" descr="External public Identifier if one exits." />
  <Parameter name="SystemId" descr="System identifier of the xml file." />
  <Parameter name="LineNo" descr="Line number where the exception occurred." />
  <Parameter name="ColumnNo" descr="Column number where the exception occerred." />
  <Parameter name="Message" descr="Reason for the XML violation to occur." />
</Exception>
<Exception name="EXC_XMLNS_VALIDATION" priority="3" code="4003" descr="XMLNS Validation Error!
                                       Valid XML Namespace is 'urn:com.ibm.mq.config'.">
  <Parameter name="InvalidXMLNS" descr="Invalid XML Namespace." />
</Exception>
<Exception name="EXC_NULL_TOPOLOGY" priority="3" code="4004" descr="Topology has not been
                                       defined!" />
<Exception name="EXC_MISSING_REFERENCE" priority="3" code="4005" descr="A reference is missing
                                       in the configuration.">
  <Parameter name="ObjType" descr="Type of object that is missing." />
  <Parameter name="Reference" descr="Missing reference." />
</Exception>
<Exception name="EXC_UNKNOWN" priority="3" code="4999" descr="An Unknown exception has
                                       occurred in the system. Please send the stack trace to
                                       arunava@us.ibm.com with the Subject as PMR:MQST and
                                       your contact details." />
</exc:ExceptionCatalog>
```

[7] **Sample location is <EEH_PATH>/samp/xml/ExcCat.MQST.xml**

The Exception Catalog for the application may be imported by running the exception utility. The **EEH_PATH** environment variable must be set to the installation path of the tool either in the environment or in the batch file or shell script. On UNIX systems it may be set up in the **.profile**. A listing of the sample[8] batch file to load the exception catalog is shown below.

```
@echo off

rem ***********************************************************************
rem * Licensed Materials - Property of IBM                                *
rem * (C) Copyright IBM Corp. 2006, 2008 All Rights Reserved.             *
rem * US Government Users Restricted Rights - Use, duplication or         *
rem * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.   *
rem ***********************************************************************


set EEH_PATH=d:/tools/eeh
call bin/win/excutil.bat -import samp/xml/ExcCat.MQST.xml -drv "com.ibm.db2.jcc.DB2Driver"
       -url "jdbc:db2://localhost:50000/EXCCAT" -usr arunava -pwd *****

echo .
pause
```

The batch file may be customized for the installation path, database URL, user id, password, etc. Keep any customized batch files, notes, etc. in either a directory outside the installation directory, or in a directory called **.user** under the installation directory to avoid any over-writing when installing future releases of the product.

To run the sample programs, the application information must be set up in the database. Any application that uses the exception catalog must be defined and linked. A listing of the sample[9] DML is shown below.

```
--
-- Licensed Materials - Property of IBM
-- (C) Copyright IBM Corp. 2009 All Rights Reserved.
-- US Government Users Restricted Rights - Use, duplication or
-- disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
--

--
-- Created January 2008
--


insert into app_cat(appid,application,owner,email,descr)
       values('ExcH','ExceptionHandler','Arunava Majumdar','arunava@us.ibm.com','Enterprise
       Exception Handler');
insert into app_cat(appid,application,owner,email,descr) values('MQST','ReadConfig','Arunava
       Majumdar','arunava@us.ibm.com','MQ Scripting Tool');

insert into exc_app_cat(catalog,version,code,appid) select catalog,version,code,'ExcH'from
       exc_cat where catalog='ExcCat.ExcH';
insert into exc_app_cat(catalog,version,code,appid) select catalog,version,code,'MQST'from
       exc_cat where catalog='ExcCat.MQST';



insert into app_header(header,descr) values('AppHdr1','Application Header 1');
insert into app_header(header,descr) values('AppHdr2','Application Header 2');
insert into app_header_parm(header,parm,descr) values('AppHdr1','ABC','Application Header 1
       Parm 1');
```

[8] **Sample location is <EEH_PATH>/import.bat**

[9] **Sample location is <EEH_PATH>/samp/dml/setup.sql**

```
insert into app_header_parm(header,parm,descr) values('AppHdr2','ABC21','Application Header 2
       Parm 1');
insert into app_header_parm(header,parm,descr) values('AppHdr2','ABC22','Application Header 2
       Parm 2');

insert into app_cat_header(appid,header) values('MQST','AppHdr1');
insert into app_cat_header(appid,header) values('MQST','AppHdr2');


insert into exc_app_cat(catalog,version,code,appid)
       values('ExcCat.ExcH','1.0.0','94005','MQST');
insert into exc_app_cat(catalog,version,code,appid)
       values('ExcCat.ExcH','1.0.0','94007','MQST');
```

7. Using the Java API:

A Java API package is provided to facilitate the building of Java-based applications to adhere to the standards and integrate into the Enterprise Exception Handling Pattern. Some of the features of the API are listed below:

- Setting up the environment
- Setting up ESB for sending exceptions
- Capturing exceptions
- Sending exceptions to the queue

The class diagram for the entire package is shown below. It is a fairly simple API to facilitate applications to setup and send exceptions based on the exceptions in the standardized format to the exception reporting queue. The exception handling daemon may be set up to service the queue. The queue may be in a cluster for load balancing reasons.

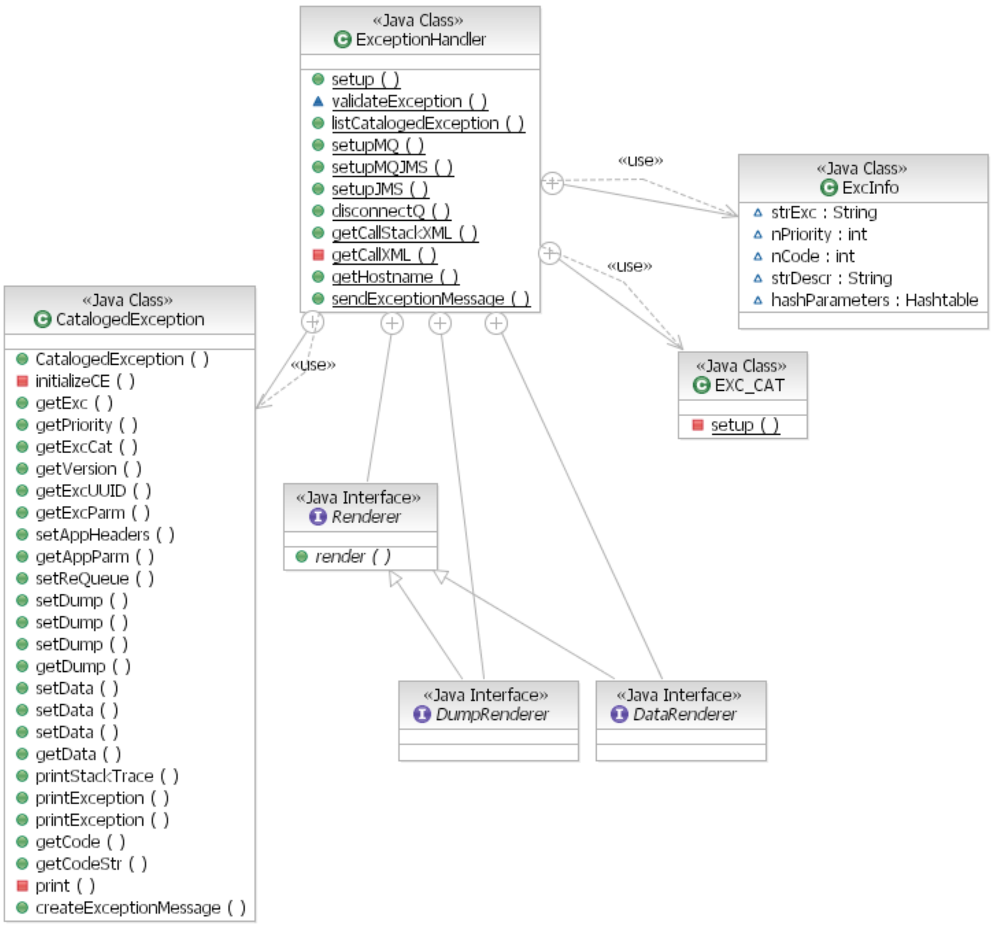


Figure 11 – Java API Class Diagram

7.1. Creating the Exception Catalog Class

To facilitate coding and capturing of runtime exceptions, an exception catalog class can be created using the exception utility. This class creates functions and ensures that the exception parameters are properly setup. The exception parameters are checked at runtime and the API throws a **CatalogedException** if the parameters supplied to the exception do not match that in the Exception Catalog. This is to ensure that all the exceptions reported to the centralized Exception Handling Daemon (**exchd**) are valid exceptions assigned to the application. All exception conditions must be tested and these should be caught as bugs in the process. The exception catalog class generation is a step forward for catching these erroneous parameters at compile time. Even after the best testing and coding efforts, if exceptions are sent to the daemon, they are catch as foreign key violations in the database and are databased with internal exception handler codes[10].

Following is a listing of the sample java class generation script[11] provided in the package.

```
@echo off

rem **********************************************************************
rem * Licensed Materials - Property of IBM                               *
rem * (C) Copyright IBM Corp. 2006, 2008 All Rights Reserved.            *
rem * US Government Users Restricted Rights - Use, duplication or        *
rem * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.  *
rem **********************************************************************


set EEH_PATH=D:/tools/eeh
call bin/win/excutil.bat -classgen Java -classdir samp/java/com/ibm/mq/exc/samp
                         -exccat samp/xml/ExcCat.MQST.xml -package com.ibm.mq.exc.samp
echo .
pause
```

[10] **Please refer to the internal Exception Catalog in Appendix I**

[11] **Sample location is <EEH_PATH>/classgen.bat**

The generated class provided in the samples[12] directory is listed partially below for reference showing one of the functions that were created for the Exception `EXC_MISSING_REFERENCE`.

```
package com.ibm.mq.exc.samp;

import java.util.Hashtable;

import com.ibm.mq.exc.ExceptionHandler;
import com.ibm.mq.exc.ExceptionHandler.CatalogedException;

public class ExcCat_MQST{

        public static final String name = "ExcCat.MQST";
        public static final String version = "1.0.0";

        /**
         * A reference is missing in the configuration.
         *
         * @param strReference Missing reference.
         * @param strObjType   Type of object that is missing.
         *
         * @return CatalogedException for named exception <B>EXC_MISSING_REFERENCE</B>
         *
         * @exception CatalogedException of the following flavours:<BR>
         *                  - EXC_CAT.CATALOG_UNDEF<BR>
         *                  - EXC_CAT.EXCEPTION_UNDEF<BR>
         *                  - EXC_CAT.PARM_INVALID<BR>
         *                  - EXC_CAT.PARM_INVALID_INTERNAL<BR>
         */
        public static CatalogedException excEXC_MISSING_REFERENCE(
                                             String strReference,
                                             String strObjType) throws CatalogedException {

                Hashtable<String,String> hashParm = new Hashtable<String,String>();
                hashParm.put("Reference",strReference);
                hashParm.put("ObjType",strObjType);

                return new CatalogedException(ExceptionHandler.PRI_ERR,
                                             "EXC_MISSING_REFERENCE",hashParm,name,version);
        }


                         • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •
```

[12] **Sample location is <EEH_PATH>/samp/java/com/ibm/mq/exc/samp/ExcCat_MQST.java**

7.2. Setting up the Environment

To setup the Exception Handling environment a call to the static method **`ExceptionHandler.setup`** ***is mandatory***. This loads the Exception Catalog into the JVM. The function should be called for all the exception catalogs that the application uses. Ideally only one Exception Catalog should be used for the exceptions generated from the application. The static call to **`ExceptionHandler.setupMQ`** is required to setup the MQ environment for the application to send messages to the centralized exception handling queue. A complete example is provided in the samples[13].

```
        private static String QMGR = "QM.HUB";
        private static String QUEUE = "Q.FAIL";

        private static String CATALOG = "ExcCat.MQST";
        private static String CAT_VER = "1.0.0";

        public static void initialize() throws CatalogedException {
                String strPath=System.getenv("EEH_PATH");
                if (strPath==null || strPath.length()==0) {
                        System.out.println("ERROR: EEH_PATH not set !!!");
                        System.out.println("Please set this environment variable to the
                                          installation path of EEH.");
                        System.exit(1);
                }

                ExceptionHandler.setup(strPath+"/samp/xml/ExcCat.MQST.xml");
                ExceptionHandler.setupMQ(QMGR,QUEUE);

        }
```

An alternative method, **`ExceptionHandler.setupExcCat`**, is provided with the Exception Handler API to load the property based Exception Catalog[14]. This is provided for compatibility and conflict aversion with software using DOM4J libraries that the Exception Handler uses to parse the XML. The recommended approach is to use the XML based Exception Catalog and all tools are based on that schema. Below is an example to load the property based catalog from the provided sample[15].

```
        private static String QMGR = "QM.HUB";
        private static String QUEUE = "Q.EEH.FAIL";

        private static String CATALOG = "ExcCat.MQST";
        private static String CAT_VER = "1.0.0";

        public static void initialize() throws CatalogedException {
                String strPath=System.getenv("EEH_PATH");
                if (strPath==null || strPath.length()==0) {
                        System.out.println("ERROR: EEH_PATH not set !!!");
                        System.out.println("Please set this environment variable to the
                                          installation path of EEH.");
                        System.exit(1);
                }

                ExceptionHandler.setupExcCat(strPath+"/samp/exccat/ExcCat.MQST.exccat",false);
                ExceptionHandler.setupMQ(QMGR,QUEUE);
        }
```

[13] **Sample location is <EEH_PATH>/samp/java/com/ibm/mq/exc/samp/ExcH_Java_01_OK.java**
[14] **Please refer to Appendix II**
[15] **Sample location is <EEH_PATH>/samp/java/com/ibm/mq/exc/samp/ExcH_Java_04_ExcCat.java**

The Exception Catalog can also be loaded statically by setting the environment variable **EEH_SETUP** either at the system level or at the user level or even in a batch or shell script to start the application. This method of catalog loading requires no coding to set up the catalog and may be used inside the Message Broker as well. The only disadvantage of doing this is control of loading specific catalogs at a given point in the code. However, for most applications this method works to the advantage of not having the call the API directly. Another advantage to the static loading is that Exception Handler automatically detects whether the catalog is XML based or property based. Multiple catalogs can also be recursively loaded by pointing the variable to a directory with multiple catalogs and subdirectories with catalogs.

The capability of the static loading is demonstrated in the following samples provided:

- **ExcH_Java_06_StaticLoading.java**
  – Single Exception Catalog loading by setting **EEH_SETUP** to the catalog file
- **ExcH_Java_07_StaticLoadAll.java**
  – Multiple Exception Catalog loading by setting **EEH_SETUP** to the catalog directory

Exception Catalogs can also be registered with the EEH system and loaded from the registry. To register a catalog, simply copy the catalog in the following directory:

EEH_PATH/.registry/exccat

The static call to **ExceptionHandler.setupRegisteredCatalogs** loads all the catalogs in the catalog registry. The sample **ExcH_Java_08_AllReg.java** demonstrates this capability.

```
private static String QMGR = "QM.HUB";
private static String QUEUE = "Q.EEH.FAIL";

private static String CATALOG = "ExcCat.MQST";
private static String CAT_VER = "1.0.0";

public static void initialize() throws CatalogedException {
        String strPath=System.getenv("EEH_PATH");
        if (strPath==null || strPath.length()==0) {
                System.out.println("ERROR: EEH_PATH not set !!!");
                System.out.println("Please set this environment variable to the
                                    installation path of EEH.");
                System.exit(1);
        }

        ExceptionHandler.setupRegisteredCatalogs();
        ExceptionHandler.setupMQ(QMGR,QUEUE);
        ExceptionHandler.setAppId("AllReg");

        new File(strPath+"/test/Exc_Java_08").mkdirs();
        ExceptionHandler.listCatalogedException(strPath+"/test/Exc_Java_08");

}
```

Java class for the exception catalog may be generated from the exception catalog as explained in 5.1 Exception Handler Utility. The utility can only be used to generate Exception Catalog Java class from an XML Catalog. The API call to **`ExceptionHandler.getExcCatInfo`** returns the **`ExcCatInfo`** object that may be used in the **`ExceptionHandler.setup`** call. This method is particularly important when the application wants to guarantee that the catalog is always loaded correctly and not having to rely on the XML file or the **`DOM4J`** libraries, e.g. products using EEH catalogs. The disadvantage of this method is that the catalog is tightly coupled with the code.

The sample **`ExcH_Java_09_ExcCatInfoLoad.java`** demonstrates this capability.

```
        private static String QMGR = "QM.HUB";
        private static String QUEUE = "Q.EEH.FAIL";

        private static String CATALOG = "ExcCat.MQST";
        private static String CAT_VER = "1.0.0";

        public static void initialize() throws CatalogedException {
                String strPath=System.getenv("EEH_PATH");
                if (strPath==null || strPath.length()==0) {
                        System.out.println("ERROR: EEH_PATH not set !!!");
                        System.out.println("Please set this environment variable to the
                                            installation path of EEH.");
                        System.exit(1);
                }

                ExceptionHandler.setup(ExcCat_MQST.getExcCatInfo(),true);
                ExceptionHandler.setupMQ(QMGR,QUEUE);
        }
```

7.3. Generating Runtime Exceptions

The following is a code snippet to demonstrate the generation of exceptions under specific conditions in the application. Optional application headers may be attached to the exception message. The application headers must be setup in the database and associated with the application. The setup DML sets up two application headers, viz. **AppHdr1** and **AppHdr2**.

```
Hashtable<String,String> hashErrorParms = new Hashtable<String,String>();
hashErrorParms.put("InvalidXMLNS","test");
CatalogedException ceInvalidXMLNS
        = new CatalogedException(ExceptionHandler.PRI_FATAL,
                "EXC_XMLNS_VALIDATION",hashErrorParms,CATALOG,CAT_VER);

Hashtable<String,Hashtable<String,String>> hashAppHdr
        = new Hashtable<String,Hashtable<String,String>>();
Hashtable<String,String> hashAppNV = new Hashtable<String,String>();
hashAppNV.put("ABC","123");
hashAppHdr.put("AppHdr1",hashAppNV);
Hashtable<String,String> hashAppNV2 = new Hashtable<String,String>();
hashAppNV2.put("ABC21","123");
hashAppNV2.put("ABC22","123");
hashAppHdr.put("AppHdr2",hashAppNV2);
ceInvalidXMLNS.setAppHeaders(hashAppHdr);

throw ceInvalidXMLNS;
```

Alternatively, the Exception Catalog class may be used for convenience and ensuring that exception parameters are properly populated.

```
CatalogedException ceInvalidXMLNS
                        = ExcCat_MQST.excEXC_XMLNS_VALIDATION("test");

Hashtable<String,Hashtable<String,String>> hashAppHdr
        = new Hashtable<String,Hashtable<String,String>>();
Hashtable<String,String> hashAppNV = new Hashtable<String,String>();
hashAppNV.put("ABC","123");
hashAppHdr.put("AppHdr1",hashAppNV);
Hashtable<String,String> hashAppNV2 = new Hashtable<String,String>();
hashAppNV2.put("ABC21","123");
hashAppNV2.put("ABC22","123");
hashAppHdr.put("AppHdr2",hashAppNV2);
ceInvalidXMLNS.setAppHeaders(hashAppHdr);

throw ceInvalidXMLNS;
```

7.4. Capturing Runtime Exceptions

Exceptions generated in the application may be caught and a **CatalogedException** thrown. The **CatalogedException** can be caught in a general catch block and reported to the exception handling system. The Java exception can also be added to the Exception Dump for future reference. The exception is base64 encoded and added to the XML. A dump may also be added as a byte array for binary dump or xml or string for character dumps.

```
try {
        FileOutputStream fos = new FileOutputStream("test.dat");
        fos.write("This is a test message.\n".getBytes());
        fos.close();
}
catch(FileNotFoundException eFNF) {
        CatalologedException eCat = ExcCat_MQTT.excFILE_NOT_FOUND("test.dat");
        eCat.setDump(eFNF);
        throw eCat;
}
catch(IOException eIO) {
        throw ExcCat_MQTT.excIO();
}
```

Some of the common exceptions may also use the overloaded constructors of the **CatalogedException** class. Please refer to the javadocs for detailed information on all the functionality of the API.

```
throw new CatalogedException(ExceptionHandler.PRI_FATAL, eFNF, "test.dat",
                              ExcCat_MQST.name, ExcCat_MQST.version);
```

7.5. Reporting Exceptions

Reporting exceptions using the API is very simple. The **printStackTrace()** function is overridden and prints the exception code and description to the standard error and the XML exception to the standard output. The **createExceptionMessage()** returns the XML formatted string and may be saved to a log file. Also the exception message may be send to a queue using the **sendMessage()** function.

```
try {

          • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •
}
catch (CatalogedException ce) {
        try {
                ExceptionHandler.sendExceptionMessage(ce);
                if (log!=null)
                        log.warn("\n"+eCE.createExceptionMessage()+"\n");
        }
        catch (CatalogedException ce2) {
                System.out.println("Exception message:");
                ce.printStackTrace();
                System.out.println("ERROR: Sending exception message!");
                ce2.printStackTrace();
        }
}
```

7.6. Disconnecting from the Middleware

If the environment is set up to send messages to a queue, the **disconnect()** function must be called before exiting the application for properly closing the open objects and disconnecting from the queue manager.

```
finally {
        ExceptionHandler.disconnectQ();
}
```

7.7. Running Sample Applications

In order to run the sample applications, the sample queues have to be created. A sample MQSC script is provided to do this and is listed below.

```
****************************************************************************
* Licensed Materials - Property of IBM                                     *
* (C) Copyright IBM Corp. 2009 All Rights Reserved.                        *
* US Government Users Restricted Rights - Use, duplication or              *
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.        *
*                                                                          *
* Created January 2008                                                     *
****************************************************************************

def ql('Q.FAIL') bothresh(3) boqname('Q.FAIL.BAK') replace
def ql('Q.FAIL.BAK') replace
def ql('Q.CMD') replace
def ql('Q.ExcH.ESQL.01') replace
def ql('Q.ExcH.Java.01') replace
```

Please make sure the sample exception catalog is setup by importing the **samp/xml/ExcCat.MQST.xml** and setting up the application information by running the DML **samp/dml/setup.sql**.

Customize the batch files and run the sample Java applications. All batch files are located in the **samp/java** directory. Below is the listing of the **ExcH_Java_01_OK.bat** file for reference.

```
@echo off

rem ***********************************************************************
rem * Licensed Materials - Property of IBM                                *
rem * (C) Copyright IBM Corp. 2006, 2008 All Rights Reserved.             *
rem * US Government Users Restricted Rights - Use, duplication or         *
rem * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.   *
rem ***********************************************************************


set EEH_PATH=d:/tools/eeh


set OLD_CLASSPATH=%CLASSPATH%
set CLASSPATH=%EEH_PATH%/lib/com.ibm.mq.exc.jar;%EEH_PATH%/lib/ext/dom4j-
     1.6.1.jar;%EEH_PATH%/lib/ext/jaxen-core.jar;%EEH_PATH%/lib/ext/jaxen-
     dom4j.jar;%EEH_PATH%/lib/ext/saxpath.jar;%CLASSPATH%
rem ---------------------------

java com.ibm.mq.exc.samp.ExcH_Java_01_OK QM.HUB

rem ---------------------------
set CLASSPATH=%OLD_CLASSPATH%
set OLD_CLASSPATH=

echo.
pause
```

8. Using the Eclipse Plug-in:

The eclipse plug-in for EEH is provided as an eclipse update site in the **%EEH_PATH%/eclipse/site** directory. The plug-in can be very easily installed from the eclipse Help menu (**Help → Software Updates → Find and Install…**). The minimum requirement for the eclipse plug-in installation is a version of eclipse 3.2 or higher (**Callisto**). All currently supported WebSphere® MQ or Message Broker products can install the plug-in, e.g. WMQ Explorer 7.0+, WMB Toolkit 6.1+, etc. For more details on installation instructions please refer to the **EEH – Getting Started** document.

In this release the plug-in provides an editor associated with the **.excdb** file extension. The file is implemented as a simple property file in this release; but is not part of the specification. It will be implemented as an XML in future releases. Password to connect to the database is encrypted and base-64 encoded for security. The file may be copied and altered but the recommended approach is to use a copy of the sample file for the database (the **%EEH_PATH%/samp/excdb/ExcDB**) and edit it with the EEH plug-in editor.

It is recommended to copy an existing sample **.excdb** file provided in **<EEH_PATH>/samp/excdb** directory. The **ExcDB.zip** file may be imported as an eclipse project interchange file. For eclipse 3.2 import the ExcDB project from the in **<EEH_PATH>/samp/excdb/ExcDB** directory as an existing project.

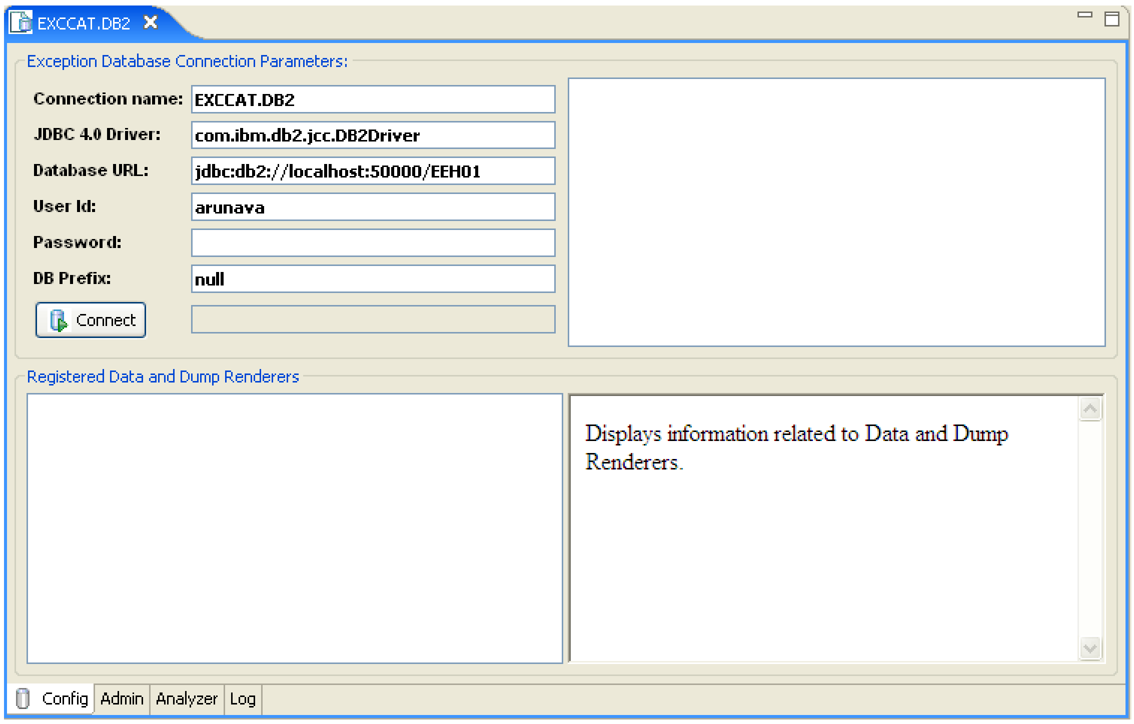
For more details on how to use the eclipse editor please refer to the ***EEH – Getting Started.doc*** chapter 3.

8.1. Configuration Page

The configuration page for the exception database editor is divided into four quadrants. The top half of the page is for database information and the bottom half for the Renderer plug-in information.

On the top left, the database JDBC connection information must be provided. On the top right, the JDBC driver jars for the database may be added. The jars are added to the service OSGi bundle for the EEH editor and the name of the package for the JDBC driver class is exported. If the class name is not provided before invoking the add database jar wizard, the driver will not be loaded. The package name is parsed and shown in a non-editable text box on the wizard.[16]

On the bottom left, the renderers may be added. It similarly adds the jars to the service OSGi bundle for the EEH editor. The jar is inspected for classes that implement the **IRenderer** interface and the **getIcon()** method is invoked to show the Icon in the Registered Renderer tree. The **getAbout()** method is invoked to display information regarding the renderer on the browser on the bottom right quadrant. HTML or text content may be displayed.


EXCCAT.DB2
Exception Database Connection Parameters:

Connection name:	EXCCAT.DB2
JDBC 4.0 Driver:	com.ibm.db2.jcc.DB2Driver
Database URL:	jdbc:db2://localhost:50000/EEH01
User Id:	arunava
Password:	
DB Prefix:	null

Connect
Registered Data and Dump Renderers

Displays information related to Data and Dump Renderers.

Config
Admin
Analyzer
Log

Figure 12 – EEH Editor: Configuration Page

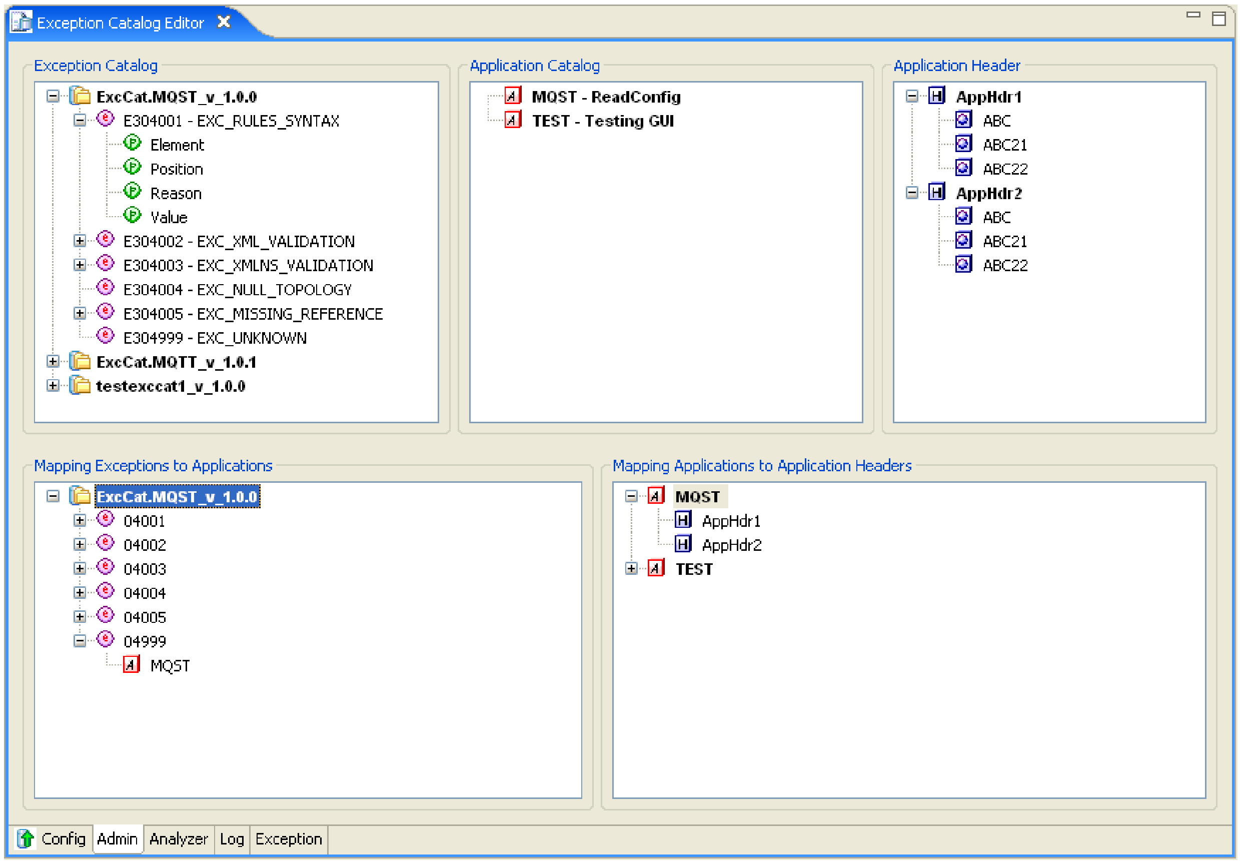
[16] **Please be sure the correct driver is being exported by the Add Database Driver Jar Wizard.**

8.2. Administration Page

The administration page is divided into two halves for displaying the catalog information from the database and the relationship between the catalogs in the database. The page contents are displayed when connected to the exception database. After disconnecting from the database, the information is not cleared. This is to save the last connection state for reference. However, none of the actions on the database would work without an explicit connection to the database. The connection state is always displayed as the icon for the Configuration page, so that it is visible from every page and does not require to explicitly go to the Configuration page just for checking the status at any point.

The top left displays the exception catalog. Catalogs may be imported/exported or added/deleted from the database through the respective wizards. Similar actions may be performed for the application catalog in the top middle and for custom application headers on the top right.

The exception catalog or the application headers may be associated with the application by dragging them to the application. The associations are displayed in the respective mapping trees at the bottom. Associations may be deleted from the mapping tree. Exceptions may be associated with an application at a catalog or at the individual exception levels.



Exception Catalog

```
⊟ ExcCat.MQST_v_1.0.0
    ⊟ E304001 - EXC_RULES_SYNTAX
        Element
        Position
        Reason
        Value
    ⊞ E304002 - EXC_XML_VALIDATION
    ⊞ E304003 - EXC_XMLNS_VALIDATION
      E304004 - EXC_NULL_TOPOLOGY
    ⊞ E304005 - EXC_MISSING_REFERENCE
      E304999 - EXC_UNKNOWN
⊞ ExcCat.MQTT_v_1.0.1
⊞ testexccat1_v_1.0.0
```

Application Catalog

```
MQST - ReadConfig
TEST - Testing GUI
```

Application Header

```
⊟ AppHdr1
    ABC
    ABC21
    ABC22
⊟ AppHdr2
    ABC
    ABC21
    ABC22
```

Mapping Exceptions to Applications

```
⊟ ExcCat.MQST_v_1.0.0
    ⊞ 04001
    ⊞ 04002
    ⊞ 04003
    ⊞ 04004
    ⊞ 04005
    ⊟ 04999
        MQST
```

Mapping Applications to Application Headers

```
⊟ MQST
    AppHdr1
    AppHdr2
⊞ TEST
```

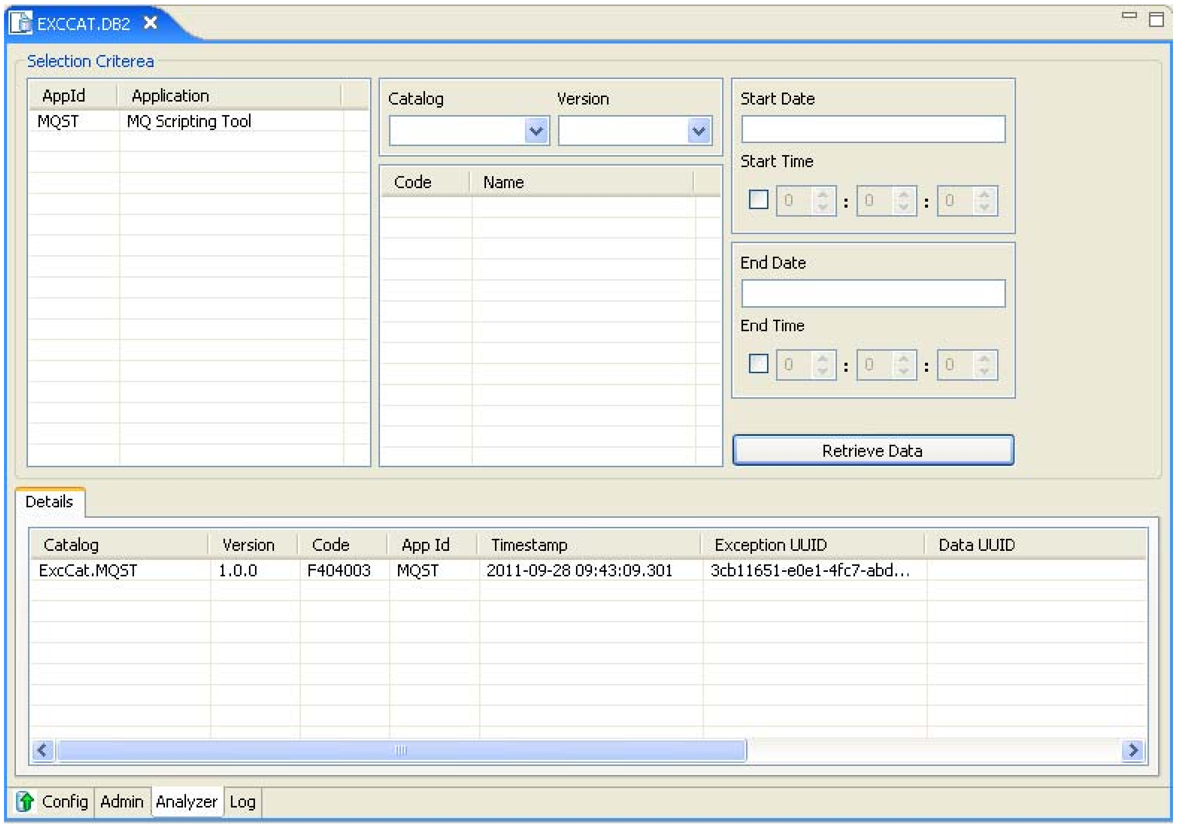
Figure 13 – EEH Editor: Administration Page

8.3. Analysis Page

The exception analysis page is horizontally divided into two parts. The top part provides the search criteria for the exception and the bottom table displays the selected exceptions from the database.

The application may be selected from the table on the left. For deselecting all applications, press the escape key. All available exception catalogs are displayed in the combo box. Only after selecting the catalog, the versions for that catalog present in the database are displayed in the catalog version combo box. Once the catalog and version are displayed, the exceptions present in the catalog for that version are displayed in the code table below. Multiple exceptions may be selected in the query. To deselect all exceptions in the table (i.e. to see al the exceptions), press the escape key. Start date/time or end date/time for the exception range may also be selected. The retrieve button runs the selection criteria and retrieves exceptions in the table below.

To view the details of the exception, double click on the exception. The details are displayed in a separate dynamically created page.



AppId	Application
MQST	MQ Scripting Tool

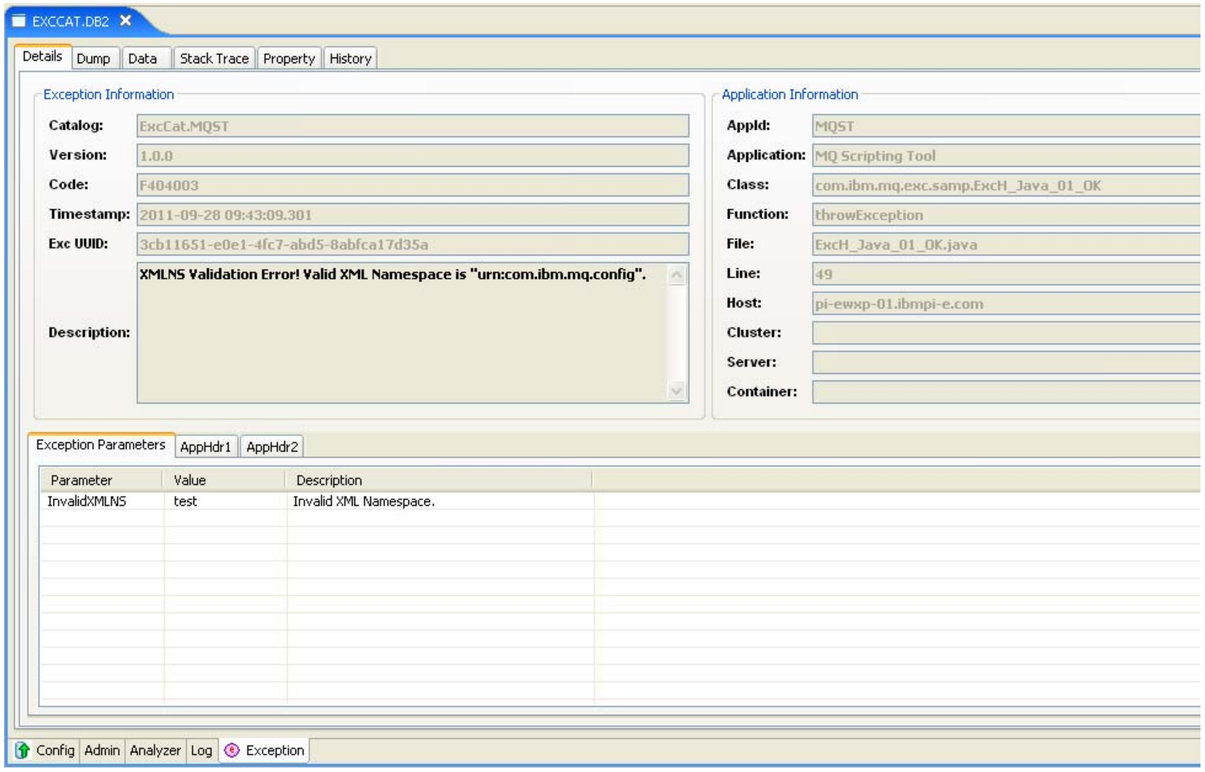
Catalog	Version	Code	App Id	Timestamp	Exception UUID	Data UUID
ExcCat.MQST	1.0.0	F404003	MQST	2011-09-28 09:43:09.301	3cb11651-e0e1-4fc7-abd...	

Figure 14 – EEH Editor: Analyzer Page

8.4. Exception Page

The exception page is dynamically created when an exception is double clicked on the displayed exception in the analyzer page. More details on the available tabs on the exception page is provided in the ***EEH – Getting Started.doc*** chapter 3, section 3.6.

The re-queuing history is maintained in the database and can be viewed from the editor. The data modification history is also designed for in this release. However, the functionality to edit data is not provided. This will be provided in the next release along with the ability to invoke any Renderer from within the editor.



Catalog:	ExcCat.MQST
Version:	1.0.0
Code:	F404003
Timestamp:	2011-09-28 09:43:09.301
Exc UUID:	3cb11651-e0e1-4fc7-abd5-8abfca17d35a
Description:	XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".

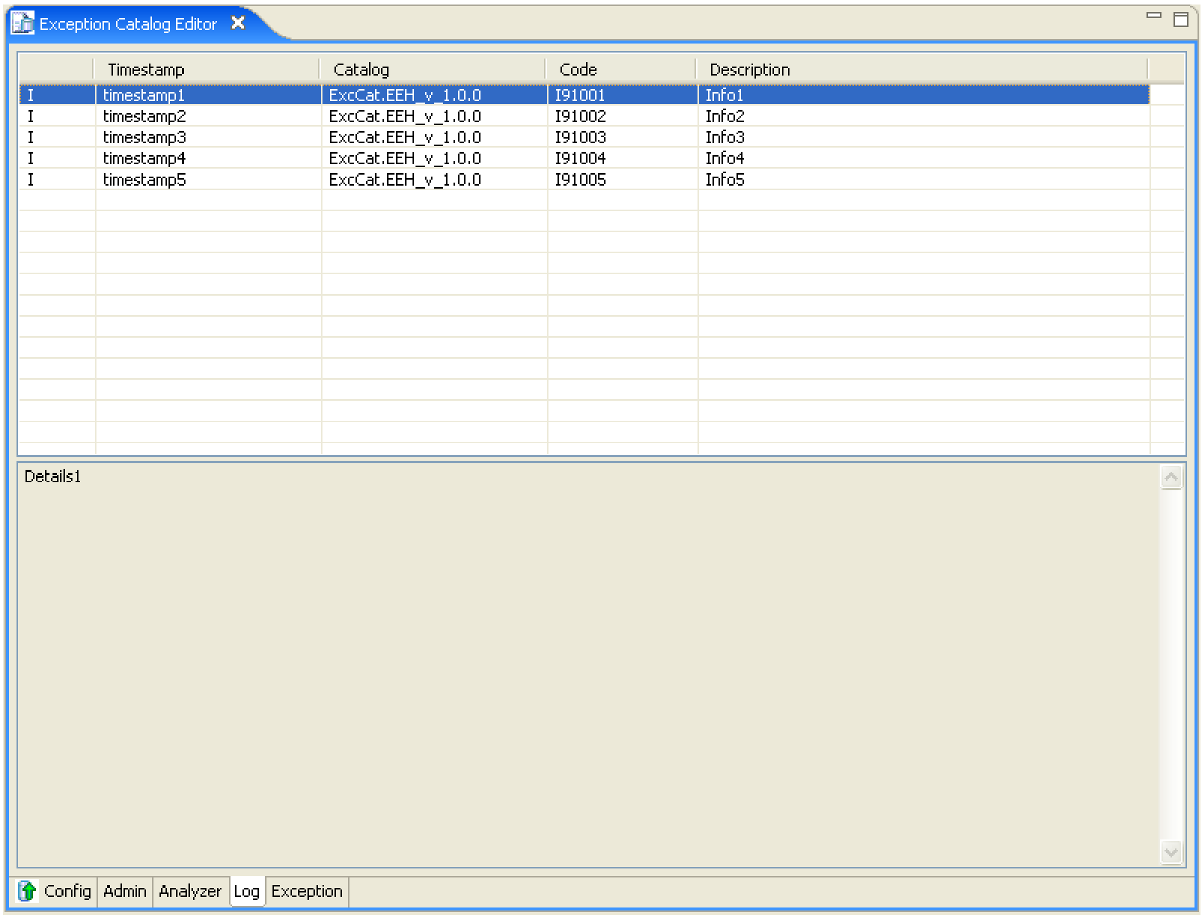
AppId:	MQST
Application:	MQ Scripting Tool
Class:	com.ibm.mq.exc.samp.ExcH_Java_01_OK
Function:	throwException
File:	ExcH_Java_01_OK.java
Line:	49
Host:	pi-ewxp-01.ibmpi-e.com
Cluster:	
Server:	
Container:	

Parameter	Value	Description
InvalidXMLNS	test	Invalid XML Namespace.

Figure 15 – EEH Editor: Exception Page

8.5. Log Page

The log page lists any exception occurred during the life-time of the editor. The editor startup errors are also stored and displayed after the editor recovers and renders itself. The details of the selected errors are displayed in the pane below. The log only exists in memory unless explicitly saved from this page. The exceptions may also be cleared at runtime.



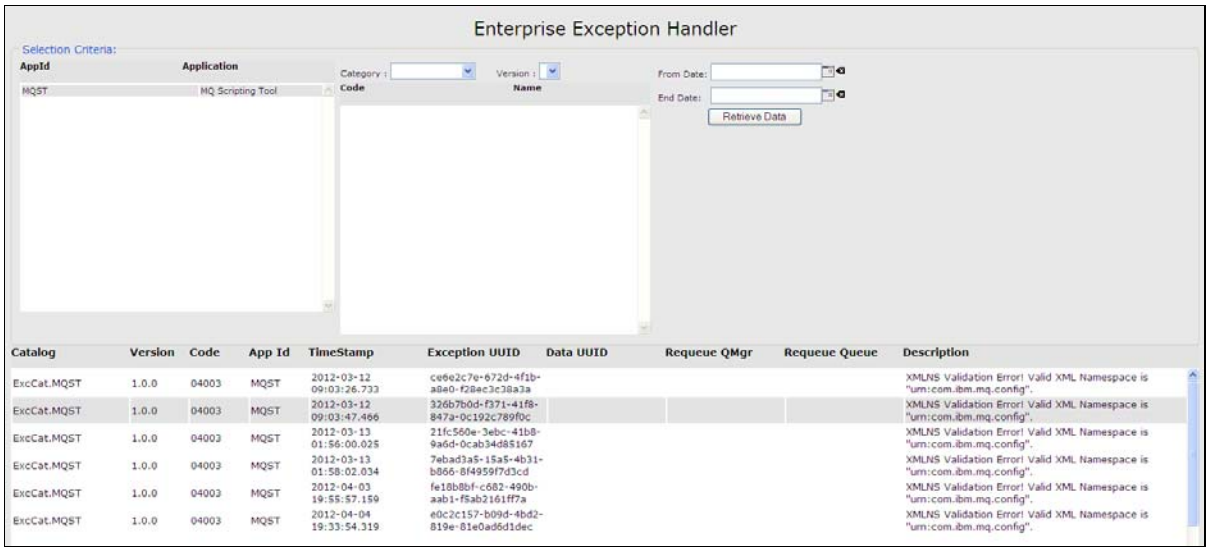
	Timestamp	Catalog	Code	Description
I	timestamp1	ExcCat.EEH_v_1.0.0	I91001	Info1
I	timestamp2	ExcCat.EEH_v_1.0.0	I91002	Info2
I	timestamp3	ExcCat.EEH_v_1.0.0	I91003	Info3
I	timestamp4	ExcCat.EEH_v_1.0.0	I91004	Info4
I	timestamp5	ExcCat.EEH_v_1.0.0	I91005	Info5

Details1

Figure 16 – EEH Editor: Log Page

9. Using the Web Analyzer:

The Web Analyzer is packaged as a web archive (**EEH.war**) file in the **%EEH_PATH%/web** directory. Please refer to the Getting Started Guide for information on how to install the application on Tomcat or the WebSphere Application Server. The Web Analyzer connects to the Exception Handling database through the JDBC driver. The driver must be configured for the JNDI entry for **jdbc/EXPDB** before deploying the application.

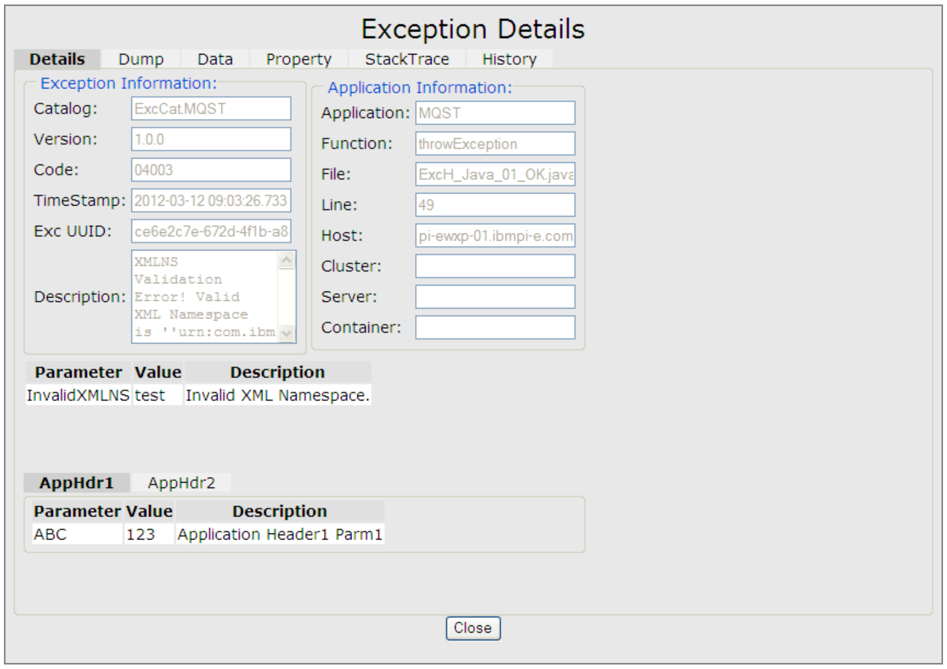


AppId	Application
MQST	MQ Scripting Tool

Code	Name

Catalog	Version	Code	App Id	TimeStamp	Exception UUID	Data UUID	Requeue QMgr	Requeue Queue	Description
ExcCat.MQST	1.0.0	04003	MQST	2012-03-12 09:03:26.733	ce6e2c7e-672d-4f1b-a8e0-f28ec3c38a3a				XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".
ExcCat.MQST	1.0.0	04003	MQST	2012-03-12 09:03:47.466	326b7b0d-f371-41f8-847a-0c192c789f0c				XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".
ExcCat.MQST	1.0.0	04003	MQST	2012-03-13 01:56:00.025	21fc560e-3ebc-41b8-9a6d-0cab34d85167				XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".
ExcCat.MQST	1.0.0	04003	MQST	2012-03-13 01:58:02.034	7ebad3a5-15a5-4b31-b866-8f4959f7d3cd				XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".
ExcCat.MQST	1.0.0	04003	MQST	2012-04-03 19:55:57.159	fe18b8bf-c682-490b-aab1-f5ab2161ff7a				XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".
ExcCat.MQST	1.0.0	04003	MQST	2012-04-04 19:33:54.319	e0c2c157-b09d-4bd2-819e-81e0ad6d1dec				XMLNS Validation Error! Valid XML Namespace is "urn:com.ibm.mq.config".

Figure 17 - EEH Web Analyzer



Exception Information:

Catalog:	ExcCat.MQST
Version:	1.0.0
Code:	04003
TimeStamp:	2012-03-12 09:03:26.733
Exc UUID:	ce6e2c7e-672d-4f1b-a8
Description:	XMLNS Validation Error! Valid XML Namespace is ''urn:com.ibm

Application Information:

Application:	MQST
Function:	throwException
File:	ExcH_Java_01_OK.java
Line:	49
Host:	pi-ewxp-01.ibmpi-e.com
Cluster:	
Server:	
Container:	

Parameter	Value	Description
InvalidXMLNS	test	Invalid XML Namespace.

Parameter	Value	Description
ABC	123	Application Header1 Parm1

Figure 18 – EEH Web Analyzer: Exception Details

The details of the exception is shown on double-clicking on the selected Exception.

10. Integration Point: Message Broker

An API is provided in the package for Message Broker integration. This facilitates coding flows in the Message Broker. The **ExceptionList** is encapsulated in the Exception Dump and can be readily sent to the centralized exception handling queue. There are two sets of API for integrating with ESQL as well as Java Compute Nodes. For the ESQL integration a sub-flow (**ESQ_Exception**) is provided as well as mappings to external Java functions that are not available in ESQL. The Java API is provided as part of the package **com.ibm.mq.exc.mb.jar**. Usage of both of these integration packages are described in more details in the following sections.

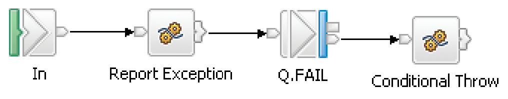


Figure 19 – ESQL_Exception Subflow

The sub-flow may be used to catch the **ExceptionList** generated by the and send the XML formatted message to the centralized exception queue. The Queue Name property of the **MQOutputNode** is promoted to the sub-flow level. Also the Environment variable `Environment.ExceptionMessage.Rethrow='Y'` determines whether the exception with the same code is thrown back to the calling message for further handling of the exception. Under most situations this is not required. The message is also sent in a non-transactional mode so that even if the message flow transaction rolls back the exception message is processed.

- **ESQL functions in Broker schema com.ibm.mq.exc.mb**
  - setup, setReQueue, setData
  - createExceptionMessage, throwCatalogedException
  - copyMessageHeaders
- **Java mappings in Broker schema com.ibm.mq.exc.mb**
  - setupExcCat, validateException, getExcCode
  - getHostname, encodeBase64, decodeBase64

The Message Broker Java API provides the **MbCatalogedException** class to encapsulate some of the Message Broker functionality, e.g. generation of the **ExceptionList**, etc. Following are the Message Broker extension functions.

- **com.ibm.mq.exc.mb.MbCatalogedException**
  - setupExcCat, validateException
  - setReQueue, setData
  - createCE, getMbException, translateMbException
  - getFirstXPath, toXML, copyMessageHeaders

10.1. Setting up the Environment

The Exception Catalog XML must be setup in the message flow similar to how a stand-alone Java application would do. However, the call to setup the catalog is slightly different in the case of the ESQL-based message flow and the Java-based message flow as stated below. Typically the **reQueue** parameter is also set if the message flow is triggered from an **MQInputNode**. The **setData** call sets up the Data section of the XML if required. When used, set up the data at the beginning of the flow since the data may be transformed from node to node and the re-queuing of data should be the original data that was sent to the queue. The data is treated as a binary data and base64 encoded.

10.1.1. ESQL

The Exception Catalog XML must be setup using the ESQL **setup** procedure call which is mapped to the Java API **ExceptionHandler.setup** function.

```
CREATE COMPUTE MODULE Initialize
	CREATE FUNCTION Main() RETURNS BOOLEAN
	BEGIN
		SET OutputRoot = InputRoot;

		create firstchild of Environment type Name name 'ExceptionMessage';
		create firstchild of Environment.ExceptionMessage type Name name
			'ExceptionHeader';
		declare ref_eMsg    reference to Environment.ExceptionMessage;
		declare ref_eHeader reference to Environment.ExceptionMessage.ExceptionHeader;

		declare EEH_PATH char getenv('EEH_PATH');

		call setup(EEH_PATH||'/samp/xml/ExcCat.MQST.xml',Environment.ExceptionMessage);

		call setReQueue(Environment.ExceptionMessage,InputRoot.MQMD);
		call setData(UUIDASCHAR,Environment.ExceptionMessage,InputRoot.BLOB.BLOB);

		RETURN TRUE;
	END;
END MODULE;
```

10.1.2. Java

The Exception Catalog XML must be setup using the Java API **ExceptionHandler.setup** function. The **ExceptionHandler.setupMQ** call should also be made to set up the MQ for sending the message to the exception queue directly through the Exception Handling API like in the case of a Java stand-alone application. However, the designer may choose to retrieve the XML formatted exception and pass it to a MQOutputNode. We recommend sending the exception message in a non-transactional mode to ensure that the message is actually sent to the queue even if the message flow transaction fails.

```
public class ExcH_Java_Initialize extends MbJavaComputeNode {

        private static String QMGR = "QM.BK";
        private static String QUEUE = "Q.FAIL";

        private static String EEH_PATH = null;
        private static boolean boolSetup = false;

        public void onDelete() {
                super.onDelete();
                ExceptionHandler.disconnectQ();
        }

        public void evaluate(MbMessageAssembly assembly) throws MbException {
                MbOutputTerminal out = getOutputTerminal("out");
                MbElement eExcMsg
                        = (MbElement)((List)assembly.getGlobalEnvironment().getRootElement()
                              .evaluateXPath("?ExceptionMessage")).get(0);
                MbElement eMQMD
                        = (MbElement)((List)assembly.getMessage().getRootElement()
                              .evaluateXPath("MQMD")).get(0);
                try {
                        if (!boolSetup) {
                                EEH_PATH = System.getenv("EEH_PATH");
                                if (EEH_PATH==null || EEH_PATH.length()==0) {
                                        Hashtable<String,String> hashPathError
                                          = new Hashtable<String,String>();
                                        hashPathError.put("Key","EEH_PATH");
                                        throw new CatalogedException(ExceptionHandler.PRI_FATAL,
                                                        "INSTALL_PATH_UNDEF",
                                                        hashPathError,"ExcCat.MQST","1.0.0");
                                }

                                ExceptionHandler.setup(EEH_PATH+"/samp/xml/ExcCat.MQST.xml");
                                ExceptionHandler.setupMQ(QMGR,QUEUE);

                                boolSetup = true;
                        }

                        MbCatalogedException.setReQueue(eExcMsg,eMQMD);
                        MbCatalogedException.setData(ExceptionHandler.generateUUID(),
                                                        eExcMsg,eMQMD);
                }
                catch (CatalogedException ce) {
                        throw MbCatalogedException.getMbException(ce,eExcMsg,getName());
                }

                out.propagate(assembly);
        }
}
```

10.2. Generating Runtime Exceptions

10.2.1. ESQL

Generating an exception in the flow is quite simple. The parameters are set in the Environment variable for all the Exception Header specific information and a User Exception is thrown.

```
declare ref_eMsg    reference to Environment.ExceptionMessage;
declare ref_eHeader reference to Environment.ExceptionMessage.ExceptionHeader;

set ref_eMsg.name='EXC_XMLNS_VALIDATION';
set ref_eHeader.version='1.0.0';
set ref_eHeader.catalog='ExcCat.MQST';
set ref_eHeader.Parameters.InvalidXMLNS='test';
set ref_eHeader.Application.class=NodeLabel;
set ref_eHeader.Application.Headers.AppHdr1.ABC='123';
set ref_eHeader.Application.Headers.AppHdr2.ABC21='123';
set ref_eHeader.Application.Headers.AppHdr2.ABC22='123';
set ref_eHeader.ReQueue.MQMD=encodeBase64(bitstream(InputRoot.MQMD));
set ref_eHeader.ReQueue.MQMD_ver=cast(InputRoot.MQMD.Version as char);
set ref_eHeader.ReQueue.Q=InputRoot.MQMD.SourceQueue;
set ref_eMsg.Data=encodeBase64(InputRoot.BLOB.BLOB);
set ref_eMsg.DataType='binary';
set ref_eMsg.UUID=UUIDASCHAR;

throw user exception severity 3 catalog 'ExcCat.MQST' message 4002;
```

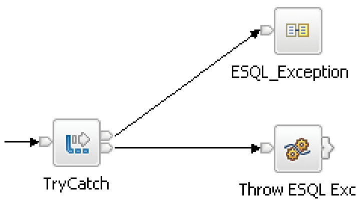
10.2.2. Java

The process of generating a **CatalogedException** from a Java Compute Node is very similar to the process of generating one in a stand-alone Java application. However, the **CatalogedException** cannot be thrown back into the message flow. The Message Broker extension API helps in converting the **CatalogedException** to an **MbException** object by using the **MbCatalogedException.*getMbException*** function call.

```
private static String CATALOG = "ExcCat.MQST";
private static String CAT_VER = "1.0.0";

public void evaluate(MbMessageAssembly assembly) throws MbException {
        MbElement eExcMsg
              = MbCatalogedException.getFirstXPath(assembly.getGlobalEnvironment()
                   .getRootElement(),"?ExceptionMessage");

        try {
                Hashtable<String,String> hashErrorParms
                     = new Hashtable<String,String>();
                hashErrorParms.put("InvalidXMLNS","test");
                CatalogedException ceInvalidXMLNS
                     = new CatalogedException(ExceptionHandler.PRI_FATAL,
                                "EXC_XMLNS_VALIDATION",hashErrorParms,CATALOG,CAT_VER);
                Hashtable<String,Hashtable<String,String>> hashAppHdr
                     = new Hashtable<String,Hashtable<String,String>>();
                Hashtable<String,String> hashAppNV = new Hashtable<String,String>();
                hashAppNV.put("ABC","123");
                hashAppHdr.put("AppHdr1",hashAppNV);
                Hashtable<String,String> hashAppNV2 = new Hashtable<String,String>();
                hashAppNV2.put("ABC21","123");
                hashAppNV2.put("ABC22","123");
                hashAppHdr.put("AppHdr2",hashAppNV2);
                ceInvalidXMLNS.setAppHeaders(hashAppHdr);

                throw MbCatalogedException
                           .getMbException(ceInvalidXMLNS,eExcMsg,getName());
        }
        catch (CatalogedException ceInternal) {
                throw MbCatalogedException
                           .getMbException(ceInternal,eExcMsg,getName());
        }
}
```

10.3. Capturing and Reporting Runtime Exceptions

10.3.1. ESQL

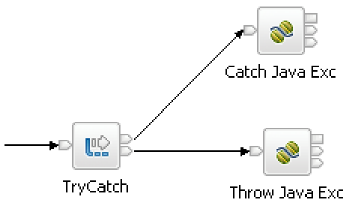


The exceptions caught in the message flow may be sent to the **ESQL_Exception** sub-flow for capturing the exception as well as sending the formatted XML message to the exception queue.

Non-user generated exceptions may be caught in the flow in the same way as user generated exceptions.

Figure 20 – Catching Exceptions with the Subflow

10.3.2. Java



The exceptions caught in the message flow may be sent to a Java Compute Node for capturing the exception and sending the formatted XML message to the exception queue. The Java code for sending the exception to the queue is listed below.

Non-user generated exceptions may be caught in the flow in the same way as user generated exceptions.

Figure 21 – Catching Exceptions with a Java Compute Node

```
public class ExcH_Java_CatchExc extends MbJavaComputeNode {

        public void evaluate(MbMessageAssembly assembly) throws MbException {
                MbOutputTerminal out = getOutputTerminal("out");
                MbElement eExcMsg
                       = MbCatalogedException.getFirstXPath(assembly.getGlobalEnvironment()
                                .getRootElement(),"?ExceptionMessage");

                try {
                        CatalogedException ce
                            = MbCatalogedException.createCE(assembly,getBroker().getName(),
                                getBroker().getQueueManagerName(),
                                getExecutionGroup().getName());

                        ExceptionHandler.sendExceptionMessage(ce);

                }
                catch (CatalogedException ce) {
                        throw MbCatalogedException.getMbException(ce,eExcMsg,getName());
                }

                out.propagate(assembly);
        }
}
```

10.4. Running Sample Applications

The sample applications for the ESQL example and the Java example are packaged under the same bar file **Exc_Samp.bar**. The bar file includes the **com.ibm.mq.exc.jar** and the **com.ibm.mq.exc.mb.jar** libraries. However, the external dependent jar files are packaged under the **ext_lib.bar**. This must be deployed to any execution group running the Exception Handler. The **com.ibm.mq.exc.jar** library must also be deployed to all the execution groups that have either ESQL-based message flows or Java-based message flows. In addition, the Java-based message flows must also deploy the **com.ibm.mq.exc.mb.jar** library.

11. Enterprise Exception Handler Forums:

IBM Services Asset: Enterprise Exception Handling: Ideas and Features

This forum discusses any new Ideas and Features that can be implemented for future releases of Enterprise Exception Handler (EEH) Pattern. This is an opportunity for not only propose a Feature Request but to involve in discussions on the usefulness of the feature. The moderators of the forum will forward any new ideas to the EEH development team and feedback from the team will be posted here. EEH development may also participate in discussions.

http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2760

IBM Services Asset: Enterprise Exception Handling: Technical Discussions

All technical discussions on the Enterprise Exception Handling (EEH) Pattern are hosted at this forum. The forum brings experts and architects and developers new to the EEH technology together to help implement EEH in the organization.

http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2761

12. Service Offering from IBM:

This Category 01 SupportPac is an IBM Services Asset for WebSphere Software and may be freely downloaded under the license agreement. Please visit the IBM Software Services Zone for WebSphere website for detailed information on services offerings and to contact a services representative.

Engage the Experts: IBM Software Services for WebSphere (ISSW)

Engage IBM to provide knowledge and expertise on the Enterprise Exception Handling (EEH) framework for your organization:

- **Enterprise Exception Handling Workshop:** A discussion forum for gathering knowledge on the scale and the complexity of the implementation of EEH. Over the 2-day workshop IBM consultant will demonstrate the capabilities and the advantages for using the framework and extensibility. High-level planning guidelines for implementation of the standard.
- **Enterprise Exception Handling Strategy and Planning:** Introducing a standard at an organization level requires planning and strategy. This is true for engineering existing applications with EEH. IBM will assist in looking at development life cycle of different application teams to coordinate the effort to provide governance around exceptions in the system and the rollout of the exception handling framework. Typically an engagement would require 4-12 weeks.
- **Enterprise Exception Handling Implementation:** IBM will provide assistance in setting up the EEH environment with the database schema and implement exception handling for a number of Java applications or message flows to provide mentoring for implementation of the pattern. Typical engagement of 3-5 weeks is recommended.
- **Enterprise Exception Handling Extensibility:** EEH provides capabilities for the implementation of plugins – Renderers, Exception Notifiers, etc. IBM can provide value added services to create plugins for specific requirements. IBM can also provide services to customize the asset for client requirements or additional integration points that are not implemented. This activity depends on the requirements and scope of the implementation. It is recommended to conduct a workshop to determine the scope of the engagement.

Conclusion:

The initial release of the Enterprise Exception Handling Pattern provides not only the standards for the integrating an enterprise-wide exception reporting system, but also provides the Java API implementation, utilities for importing and exporting exception catalogs, generation of Java class for the catalog, integration extensions for the IBM WebSphere® Message Broker and an exception handling daemon out of the box. In the future releases of the package we will be publishing tools to help analyze exceptions in the database and administration of exception and application catalogs. Please feel free to contact the author for any suggestions you might have for future enhancement of the product.

Appendix I

```xml
<?xml version="1.0" encoding="UTF-8"?>
<exc:ExceptionCatalog name="ExcCat.ExcH" version="1.0.0" appid="EEH"
        expiry="604800" expiredQueue="Q.EEH.EXPIRED"
        descr="Exception Catalog for MQST Application"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:com.ibm.mq.exc ../schema/ExceptionCatalog.xsd"
        xmlns:exc="urn:com.ibm.mq.exc">

<Exception name="FILE_NOT_FOUND" priority="3" code="91001" descr="File not found Error!">
        <Parameter name="File" descr="Filename and path of the file that could not be
                   located."/>
</Exception>

<Exception name="IO" priority="3" code="91002" descr="I/O Exception generated at runtime;
           please refer to the exception dump!"/>

<Exception name="CLASS_UNDEF" priority="3" code="91003" descr="Class could not be found!">
        <Parameter name="Class" descr="Class name that could not be located."/>
</Exception>

<Exception name="91004" priority="3" code="91004" descr="Host could not be determined!"/>

<Exception name="XML_VALIDATION" priority="3" code="94001" descr="XML Validation Error!">
        <Parameter name="ColumnNo" descr="Column number where the exception occerred."/>
        <Parameter name="LineNo" descr="Line number where the exception occurred."/>
        <Parameter name="Message" descr="Reason for the XML violation to occur."/>
        <Parameter name="PublicId" descr="External public Identifier if one exits."/>
        <Parameter name="SystemId" descr="System identifier of the xml file."/>
</Exception>

<Exception name="XMLNS_VALIDATION" priority="3" code="94002" descr="XMLNS Validation Error!
           Valid XML Namespace is 'urn:com.ibm.mq.exc'.">
        <Parameter name="InvalidXMLNS" descr="Invalid XML Namespace."/>
</Exception>

<Exception name="CATALOG_NULL" priority="3" code="94003" descr="No Exception Catalog has not
been defined!"/>

<Exception name="CATALOG_REDEF" priority="3" code="94004" descr="Exception Catalog is being
           redefined!">
        <Parameter name="ExcCat" descr="Exception Catalog."/>
</Exception>

<Exception name="CATALOG_UNDEF" priority="3" code="94005" descr="Exception Catalog is not
           defined!">
        <Parameter name="CallStack" descr="Call Stack of the original Exception."/>
        <Parameter name="ExcCat" descr="Exception Catalog."/>
        <Parameter name="ExcCatVer" descr="Exception Catalog Vaersion."/>
</Exception>

<Exception name="APPID_REDEF" priority="3" code="94006" descr="Application Identifier is being
           redefined!">
        <Parameter name="AppId" descr="Application Identifier defined."/>
        <Parameter name="AppId_New" descr="New Application Identifier conflict."/>
</Exception>

<Exception name="EXCEPTION_UNDEF" priority="3" code="94007" descr="Exception is not defined in
           the catalog!">
        <Parameter name="CallStack" descr="Call Stack of the original Exception."/>
        <Parameter name="Exc" descr="Exception not found in the catalog."/>
        <Parameter name="ExcCat" descr="Exception Catalog."/>
        <Parameter name="ExcCatVer" descr="Exception Catalog Vaersion."/>
</Exception>
```

```
<Exception name="PARM_INVALID" priority="3" code="94008" descr="Exception parameter is not
           valid for the Exception in the catalog!">
       <Parameter name="CallStack" descr="Call Stack of the original Exception."/>
       <Parameter name="Exc" descr="Exception found in the catalog."/>
       <Parameter name="ExcCat" descr="Exception Catalog."/>
       <Parameter name="Parm" descr="Exception parameter not valid for the Exception."/>
</Exception>

<Exception name="QMGR_CONN_REDEF" priority="3" code="94009" descr="Queue Manager connection
           redefinition; one has already been defined!">
       <Parameter name="QMgr" descr="Queue Manager being redefined."/>
</Exception>

<Exception name="REQUEUE_NOT_SET" priority="3" code="94010" descr="Re-Queue queue name has not
           been defined!"/>

<Exception name="QUEUE_NOT_SET" priority="3" code="94011" descr="Queue name has not been
           defined!"/>

<Exception name="DUMP_REDEF" priority="3" code="94012" descr="Exception Dump redefinition; one
           has already been defined!">
       <Parameter name="DumpType" descr="The type of exception dump already set."/>
</Exception>

<Exception name="DUMP_UNSUPPORTED" priority="3" code="94013" descr="Exception Dump type is not
           supported! Supported types are binary, string, xml, java:Exception,
           java:DumpRenderer.">
       <Parameter name="DumpType" descr="The type of exception dump provided."/>
</Exception>

<Exception name="DATA_REDEF" priority="3" code="94014" descr="Data format type redefinition;
           one has already been defined!">
       <Parameter name="DataType" descr="The type of data format already set."/>
</Exception>

<Exception name="DATA_UNSUPPORTED" priority="3" code="94015" descr="Exception Data format type
           is not supported! Supported types are binary, string, xml, java:MQMessage,
           java:JMSMessage, java:DataRenderer.">
       <Parameter name="DataType" descr="The type of Data format provided."/>
</Exception>

<Exception name="PARM_INVALID_INTERNAL" priority="4" code="94998" descr="Exception parameter
           is not valid for the Exception in the catalog ExcCat.ExcH ! This is an internal
           ExceptionHandler fatal error and is caught to avoid infinite recursive exception
           invokations.">
       <Parameter name="CallStack" descr="Call Stack of the original Exception."/>
       <Parameter name="Exc" descr="Exception found in the catalog."/>
       <Parameter name="Parm" descr="Exception parameter not valid for the Exception."/>
</Exception>

<Exception name="UNKNOWN" priority="4" code="94999" descr="An Unknown exception has occurred
           in the system. Please send the stack trace to arunava@us.ibm.com with the Subject
           as PMR:ExceptionHandler and your contact details.">
       <Parameter name="CallStack" descr="XML format of the call stack when the Exception was
                   generated."/>
</Exception>

<Exception name="WMQ" priority="3" code="95001" descr="WebSphere MQ error!">
       <Parameter name="Reason" descr="Reason code returned form WMQ."/>
</Exception>

<Exception name="JMS" priority="3" code="95002" descr="Java Messaging Service error!">
       <Parameter name="ErrorCode" descr="Error code returned form JMS vendor."/>
</Exception>
```

```
<Exception name="WMB" priority="3" code="95003" descr="ExceptionList object captured! This is
           a generic Exception caught inside the Message Flow and not intended to be used for
           User Exceptions that should be catagorized and analyzed.">
       <Parameter name="Catalog" descr="NLS message catalog name."/>
       <Parameter name="ExceptionType" descr="The type of Exception that was generated. It is
                  of the following types - RecoverableException, ParserException,
                  ConversionException, DatabaseException, UserException, SecurityException,
                  CastException, MessageException, SqlException, SocketException,
                  SocketTimeoutException."/>
       <Parameter name="File" descr="C++ source file name."/>
       <Parameter name="Function" descr="C++ source function name."/>
       <Parameter name="Line" descr="C++ source file line number."/>
       <Parameter name="Name" descr="Source object name."/>
       <Parameter name="Number" descr="NLS message number."/>
       <Parameter name="Severity" descr="1 = information, 2 = warning, 3 = error."/>
       <Parameter name="Source" descr="Source object label."/>
       <Parameter name="Text" descr="Additional text."/>
       <Parameter name="Type" descr="Source object type."/>
</Exception>

<Exception name="SQL" priority="3" code="96001" descr="Database SQL error!">
       <Parameter name="SQLCODE" descr="SQLCODE for the vender specific database error."/>
       <Parameter name="SQLMSG" descr="Message for the database error."/>
       <Parameter name="SQLSTATE" descr="SQLSTATE for the database error."/>
</Exception>

</exc:ExceptionCatalog>
```

Appendix II

```
exccat.name=ExcCat.MQST
exccat.version=1.0.0
exccat.expiry=604800
exccat.expiredQueue=Q.EEH.EXPIRED
exccat.descr=Exception Catalog for MQST Application
exccat.appid=MQST

exc.EXC_RULES_SYNTAX.priority=3
exc.EXC_RULES_SYNTAX.code=04001
exc.EXC_RULES_SYNTAX.descr=Rules Syntax Error!
exc.EXC_RULES_SYNTAX.Element.descr=Element at which the syntax is incorrect.
exc.EXC_RULES_SYNTAX.Value.descr=Value of the element containing incorrect syntax.
exc.EXC_RULES_SYNTAX.Position.descr=Position where the syntax violation was detected in the
element value.
exc.EXC_RULES_SYNTAX.Reason.descr=Reason for the syantax violation to occur.

exc.EXC_XML_VALIDATION.priority=3
exc.EXC_XML_VALIDATION.code=04002
exc.EXC_XML_VALIDATION.descr=XML Validation Error!
exc.EXC_XML_VALIDATION.PublicId.descr=External public Identifier if one exits.
exc.EXC_XML_VALIDATION.SystemId.descr=System identifier of the xml file.
exc.EXC_XML_VALIDATION.LineNo.descr=Line number where the exception occurred.
exc.EXC_XML_VALIDATION.ColumnNo.descr=Column number where the exception occerred.
exc.EXC_XML_VALIDATION.Message.descr=Reason for the XML violation to occur.

exc.EXC_XMLNS_VALIDATION.priority=3
exc.EXC_XMLNS_VALIDATION.code=04003
exc.EXC_XMLNS_VALIDATION.descr=XMLNS Validation Error! Valid XML Namespace is
'urn:com.ibm.mq.config'.
exc.EXC_XMLNS_VALIDATION.InvalidXMLNS.descr=Invalid XML Namespace.

exc.EXC_NULL_TOPOLOGY.priority=3
exc.EXC_NULL_TOPOLOGY.code=04004
exc.EXC_NULL_TOPOLOGY.descr=Topology has not been defined!

exc.EXC_MISSING_REFERENCE.priority=3
exc.EXC_MISSING_REFERENCE.code=04005
exc.EXC_MISSING_REFERENCE.descr=A reference is missing in the configuration.
exc.EXC_MISSING_REFERENCE.ObjType.descr=Type of object that is missing.
exc.EXC_MISSING_REFERENCE.Reference.descr=Missing reference.

exc.EXC_UNKNOWN.priority=3
exc.EXC_UNKNOWN.code=04999
exc.EXC_UNKNOWN.descr=An Unknown exception has occurred in the system. Please send the stack
trace to arunava@us.ibm.com with the Subject as PMR:MQST and your contact details.
```

Bibliography:

1.	WMQ Support Pac md08 – WebSphere MQ Network Design Notation http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006700
2.	UML 2.0 Superstructure specification http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/
3.	The Unified Modeling Language Reference Manual by James Rambaugh, Ivar Jacobson and Grady Booch http://www.ibm.com/developerworks/rational/library/5822.html
4.	WebSphere MQ InfoCenter http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp
5.	WebSphere Message Broker InfoCenter http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r1m0/index.jsp http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp
6.	WMQ Support Pac ms03 – Save Queue Manager http://www-01.ibm.com/support/docview.wss?uid=swg24000673
7.	WMQ Support Pac ms0e – MQ Administration Wrapper http://www-01.ibm.com/support/docview.wss?uid=swg24000686
8.	IBM Alphaworks performance harness http://www.alphaworks.ibm.com/tech/perfharness
9.	IBM ESB Overview http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esbbenefits