



## Enterprise Exception Handling

*Arunava (Ron) Majumdar*

Sr. IT Specialist

Software Services for WebSphere (SE Region)

[arunava@us.ibm.com](mailto:arunava@us.ibm.com)

IBM

# Contents

<i>Modification History</i> .....	4
<i>Legal Disclaimer</i> .....	5
<i>Acknowledgement</i> .....	6
<i>Scope of the Document</i> .....	7
<b>1. Introduction to Exceptions</b> .....	<b>8</b>
1.1. What is an exception?.....	8
1.2. Exception Object concepts.....	8
1.3. Importance of exception handling.....	8
1.4. Importance of common exception handling for the organization.....	9
1.5. Importance of cataloging exceptions.....	9
1.6. Designing exceptions.....	9
<b>2. Enterprise Exception Handling Concepts</b> .....	<b>10</b>
2.1. Standardization of the Exception format.....	10
2.2. Exception reporting through the ESB.....	11
2.3. Notification through the Monitoring System.....	11
2.4. Centralization and Statistical Analysis of Exceptions.....	12
2.5. Testing Exceptions Conditions.....	12
<b>3. Design Pattern – Enterprise Exception Handling</b> .....	<b>13</b>
3.1. Architecture.....	14
3.2. Message Specifications.....	16
3.3. Database schema Specifications.....	24
<b>4. Exception Handler Runtime and Utilities</b> .....	<b>29</b>
4.1. Exception Handler Utility.....	29
4.2. Exception Handler Command.....	32
4.3. Exception Handler Daemon.....	33
<b>5. Setting up the Exception Catalog</b> .....	<b>35</b>
<b>6. Using the Java API</b> .....	<b>38</b>
6.1. Creating the Exception Catalog Class.....	39
6.2. Setting up the Environment.....	41
6.3. Generating Runtime Exceptions.....	42
6.4. Capturing Runtime Exceptions.....	43
6.5. Reporting Exceptions.....	44
6.6. Disconnecting from the Middleware.....	44
6.7. Running Sample Applications.....	45
<b>7. Integration Point: Message Broker</b> .....	<b>46</b>
7.1. Setting up the Environment.....	47
7.1.1. ESQ.....	47
7.1.2. Java.....	48
7.2. Generating Runtime Exceptions.....	49
7.2.1. ESQ.....	49
7.2.2. Java.....	50
7.3. Capturing and Reporting Runtime Exceptions.....	51
7.3.1. ESQ.....	51
7.3.2. Java.....	51
7.4. Running Sample Applications.....	52

*Conclusion:* ..... 53  
*Appendix I:*..... 54  
*Bibliography:* ..... 57

## **Table of Figures**

*Figure 1 - Common Exception Handling Process*..... 14  
*Figure 2 – Message Specification for Exception Data*..... 17  
*Figure 3 – Message Specification for Exception Data – Application section (2.1)*..... 18  
*Figure 4 – Message Specification for Exception Catalog*..... 22  
*Figure 5 –Message Specification for Command Message*..... 23  
*Figure 6 - Exception Database Schema*..... 24  
*Figure 7 – Java API Class Diagram* ..... 38  
*Figure 8 – ESQL\_Exception Subflow*..... 46  
*Figure 9 – Catching Exceptions with the Subflow*..... 51  
*Figure 10 – Catching Exceptions with a Java Compute Node*..... 51

## Modification History

Date	Version	Author	Description
06/23/2009	0.2.0	Arunava Majumdar	Final release for Support Pac id08

## Legal Disclaimer:

*Information provided has been developed as a collection of the experiences of technical services professionals over a wide variety of customer and internal IBM environments, and may be limited in application to those specific hardware and software products and levels*

*The information contained in this document has not been submitted to any formal IBM test. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk, and in some environments may not achieve all the benefits described.*

*This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.*

*IBM may not offer the products, services, or feature discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.*

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

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

*Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials of this IBM product and use of those Web sites is at your own risk.*

*Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.*

*All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice and represent goals and objectives only.*

*All prices shown are IBM's suggested list prices and are subject to change without notice. Dealer prices may vary.*

*Any performance data contained in this document was determined in a controlled environment. Therefore the results obtained in other operating environments may vary significantly. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee that these measurements will be the same on generally available systems. Some measurements quoted in the document may have been estimated through extrapolation. Actual results may vary. Users of this presentation should verify the applicable for their specific environment.*

## COPYRIGHT LICENSE:

*This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purpose of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platforms for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.*

## 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.

**This product includes software developed by the DOM4J Project (<http://www.dom4j.org/>).**  
**This product includes software developed by the SAXPath Project (<http://www.saxpath.org/>).**  
**This product includes software developed by the JAXEN Project (<http://jaxen.codehaus.org/>).**

## 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.

# 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 exists 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 to be extensible to integrate with monitoring systems so that the same monitoring mechanism may be used to 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 built 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 built around it.

---

## **2.4. Centralization and Statistical Analysis of Exceptions**

---

The idea behind the Enterprise Exception Handling 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. 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

### 3.1. 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

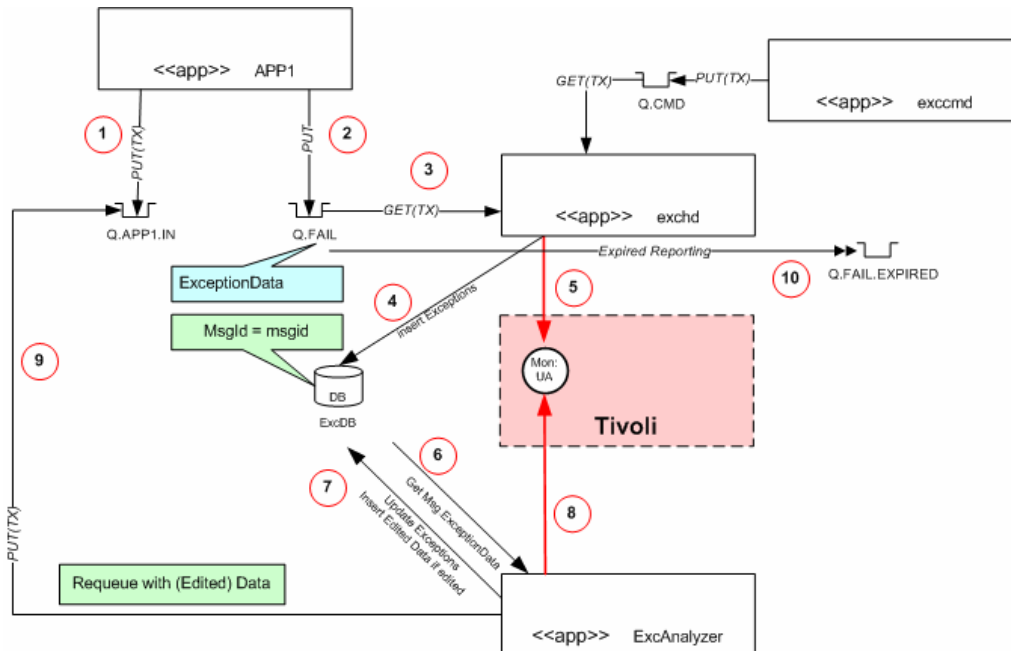


Figure 1 - Common Exception Handling Process

The diagram (Figure 1 - 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 loses 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

---

## 3.2. 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 2 – Message Specification for Exception Data).



Message Specification Diagram – ExceptionMessage

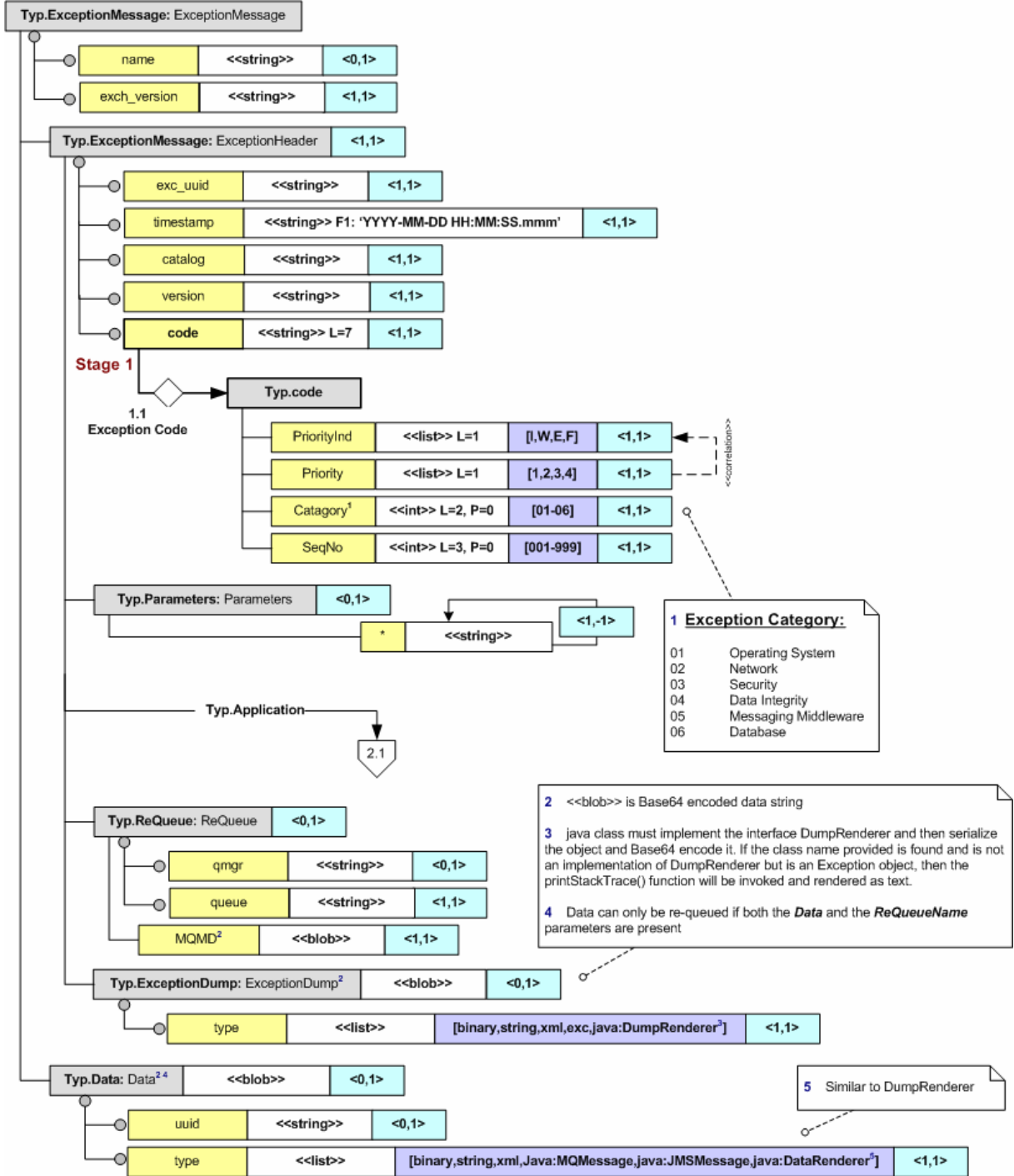


Figure 2 – Message Specification for Exception Data

Message Specification Diagram – ExceptionMessage (2.1)

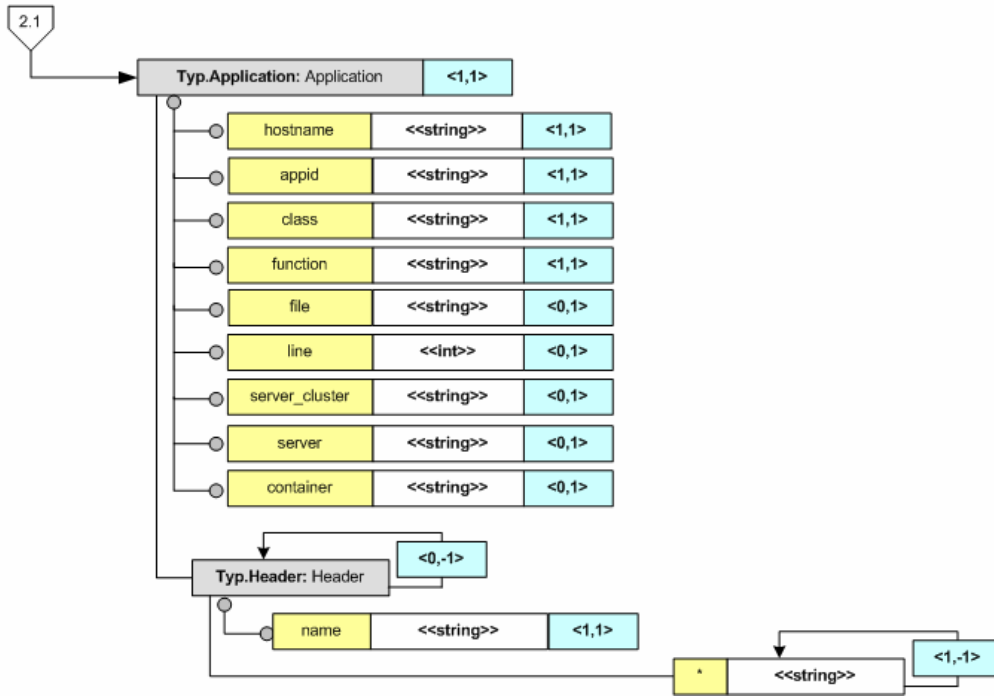


Figure 3 – Message Specification for Exception Data – Application section (2.1)

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

<i>Field</i>	<i>Description</i>
<b>exch_version</b>	The version of the Exception Header. The Version for the current release is <b>'1.0.0'</b> .
<b>name</b>	Exception variable name for reference.
<b>exc_uuid</b>	UUID generated for the exception instance.
<b>timestamp</b>	The timestamp when the Exception was caught. The format of the timestamp is a string as <b>'YYYY-MM-DD HH:MM:SS'</b> .
<b>catalog</b>	Message Catalog is the common repository of all error messages for the subsystem. This name should always be <b>'ExcCat.HSBC'</b> for the each project. This assures the existence of other projects without Exception Code clashes between the different subsystems
<b>version</b>	Version of the Exception Catalog.
<b>code</b>	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.
<b>- PriorityInd</b>	The Priority Indicator is a one character field corresponding to the Priority. It is one of the values <b>'I', 'W', 'E', or 'F'</b> .
<b>- Priority</b>	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
<b>- Catagory</b>	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.
<b>- SeqNo</b>	This is a serial number for the Exception in the class and category.
<b>Parameters</b>	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.

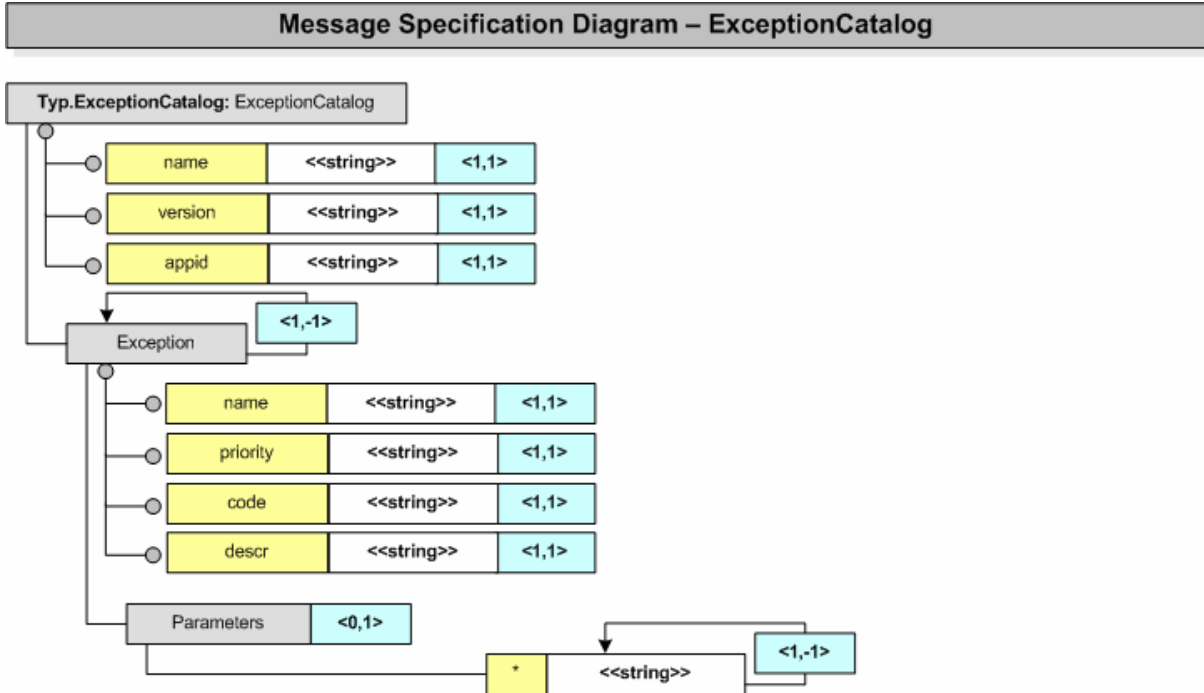
<b>Application</b>	Section for application data
- <b>hostname</b>	The host machine name or the static IP address of the machine where the Exception occurred.
- <b>appid</b>	The application identifier of the program that generated the Exception condition and sent out the report. In case of the message broker this field will contain the name of the Message Flow's application id.
- <b>class</b>	The class that generated the Exception
- <b>function</b>	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.
- <b>file</b>	File name where the exception was detected.
- <b>line</b>	Line number where the exception was detected.
- <b>server_cluster</b>	Cluster name the server hosting the application belongs to. E.g. Cell for WebSphere® Application Server, etc.
- <b>server</b>	Server name where the application is hosted. E.g. J2EE Application Server name, WebSphere® Message Broker name
- <b>container</b>	Container name inside the server hosting the application. E.g. Container name for the J2EE Application server, Execution Group for the Message Broker, etc.
- <b>Header</b>	Application Header Section will contain 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.
-- <b>name</b>	Name of the application header
<b>ReQueue</b>	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.
- <b>qmgr</b>	Optional name of the queue manager to be re-queued on
- <b>queue</b>	Name of the queue to be re-queued on
- <b>MQMD</b>	Message Descriptor of the original message for re-queuing. The binary structure is Base64 encoded.
<b>ExceptionDump</b>	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.
- <b>type</b>	The type of exception dump may be of the following flavors: <b>binary</b> – binary data dump <b>string</b> – any string type that may provide exception information <b>xml</b> – exception dumps represented in XML form excluding the XML declaration <b>exc</b> – 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. <b>java:&lt;Class&gt;</b> – serialized and Base64 encoded java class implementation of the interface DumpRenderer or an Exception class object.
<b>Data</b>	This is an optional field for the original message being processed at the time the exception occurred. The Data field is always Base64 encoded if present.
- <b>uuid</b>	This may be the MQMD.MsgId
- <b>type</b>	The type of exception dump may be of the following flavors: <b>binary</b> – binary data dump <b>string</b> – any string type that may provide exception information <b>xml</b> – exception dumps represented in XML form excluding the XML declaration <b>java:&lt;Class&gt;</b> – serialized and Base64 encoded java class implementation of the interface DataRenderer. <b>java:MQMessage</b> or <b>java:JMSMessage</b> – 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
      BW1zZ0lkSQAKbnVtSW5zZXJ0c0kACnJIYXNvbKkVZGVMAAdpbnNlcXQdAASTGphdmEvdGFuZy9TdHJpbmc7TAAHaW5zZ
      XJ0MnEafgABTAALb3N0ck1lc3NhZ2VxAH4AAUwAB3Y3bXNnSWRxAH4AAxhyABNqYXZlLmxbmcmcuRXhjZXB0aW9uO0PofPh
      o7HMQCAAB4cgATamF2YS5sYW5nLIRocm93YWJsZdXGNSc5d7jLAWADTAAFY2F1c2V0ABVMamF2YS9sYW5nL1Rocm93
      YWJsZTtMAA1kZXRhaWxNZXNzYWdlcQB+AAFBAApzdGFja1RyYWNldAAeW0xqYXZlL2xhbmcvU3RhY2tUcmFjZUUsZWV1bn
      nQ7eHBwchVvAB5bTGphdmEubGFuZy5TdGFja1RyYWNIRWxlbWVudDsCRio8PP0iOQIAAHhWAAAAACXNyABtqYXZlLmxb
      mcmcuU3RhY2tUcmFjZUUsZWV1bnRhCcWajbdbhQIABEkACmXpbmV0dW1iZXJMAA5kZWNSYXJpbmdDbGFzc3EafgABTAAIZm
      lsZU5hbWVxAH4AAUwACm1ldGhvZE5hbWVxAH4AAxhwAAABPHQAGGNvbS5pYm0ubXEuTVFEZXN0aW5hdGlvbnQAek1R
      RGVzdGluYXRpb24uamF2YXQABG9wZW5zcQB+AAkAAAEEdAASY29tLmlibS5tcS5NUUVF1ZXVldAAmTVFRdWV1ZS5qYXZl
      dAAGPGluaXQ+c3EafgAJAAAKrnQAGWNvbS5pYm0ubXEuTVFRdWV1ZU1hbmFnZXJ0ABNNUVF1ZXVITWFuYWdlci5qYXZl
      hdAALYWNjZXNzUXVldWVzcQB+AAkAAArKdAAZY29tLmlibS5tcS5NUUVF1ZXVITWFuYWdlcnQAEE01RUXVldWVNYW5hZ2V2Y
      LmphdmF0AAthY2Nlc3NRdWV1ZXNxAH4ACQAAAABJ0AB5jb20uaWJtLm1xLnRlc3QuZHMuUXVldWVtV3VyY2V0ABBRdWV1
      ZVNvdXJjZS5qYXZhdAAAGFuZGxiTVFDb25uc3EafgAJAAAKHQAHmNvbS5pYm0ubXEuTVFEZXN0aW5hdGlvbnQAek1R
      XQAEFF1ZXVlU291cmNlLmphdmF0AAhzZW5kRGF0YXNxAH4ACQAAAFR0ABljb20uaWJtLm1xLnRlc3QuUmVndWxhdG9yd
      AAOUmVndWxhdG9yLmphdmF0AANydW5zcQB+AAkAAAIpDAAVamF2YS51dGlsLIRpbWVvGhyZWZkdAAKVGlTZXluamF2Y
      XQACG1haW5Mb29wc3EafgAJAAAB3XQAFWphdmEudXRpbC5UaW1lclRocmVhZHQACIRpbWVvLmphdmF0AANydW54AA
      AAAgAAAAAAAAAAAAIJBWdAAStVFKRTAwMTogQ29tcGxldGlvbiBD2RIICyJywgUmVhc29uLm1j50AAAA=

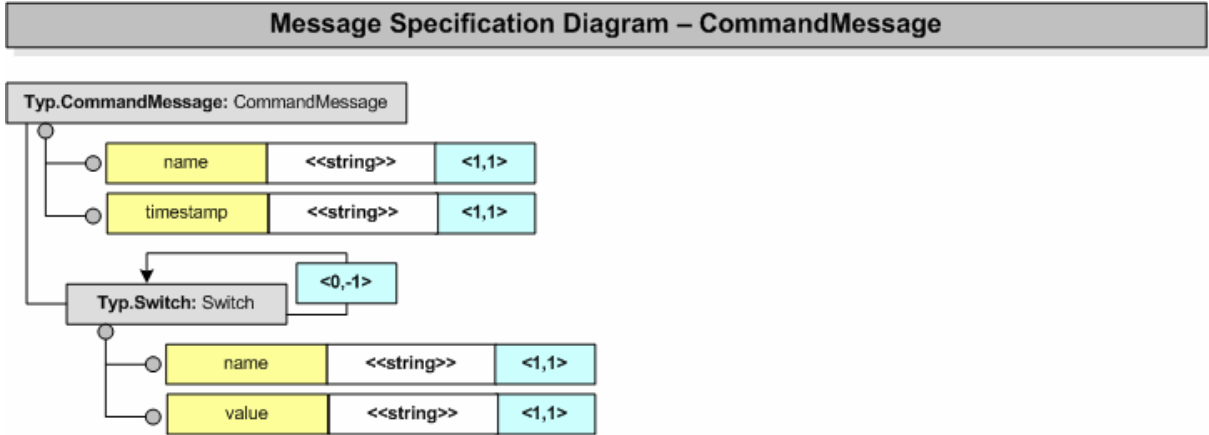
    </ExceptionDump>
  </ExceptionHeader>
</exc:ExceptionMessage>
```



**Figure 4 – 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.

<b>Field</b>	<b>Description</b>
<b>name</b>	The Exception Catalog name.
<b>version</b>	The Exception Catalog version.
<b>appid</b>	The application identifier that uses the exception catalog.
<b>Exception</b>	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, SQLERRORTXT. It is represented as a name-value pair and is fixed for the exception code.



**Figure 5 –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.

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

### 3.3. Database schema Specifications

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

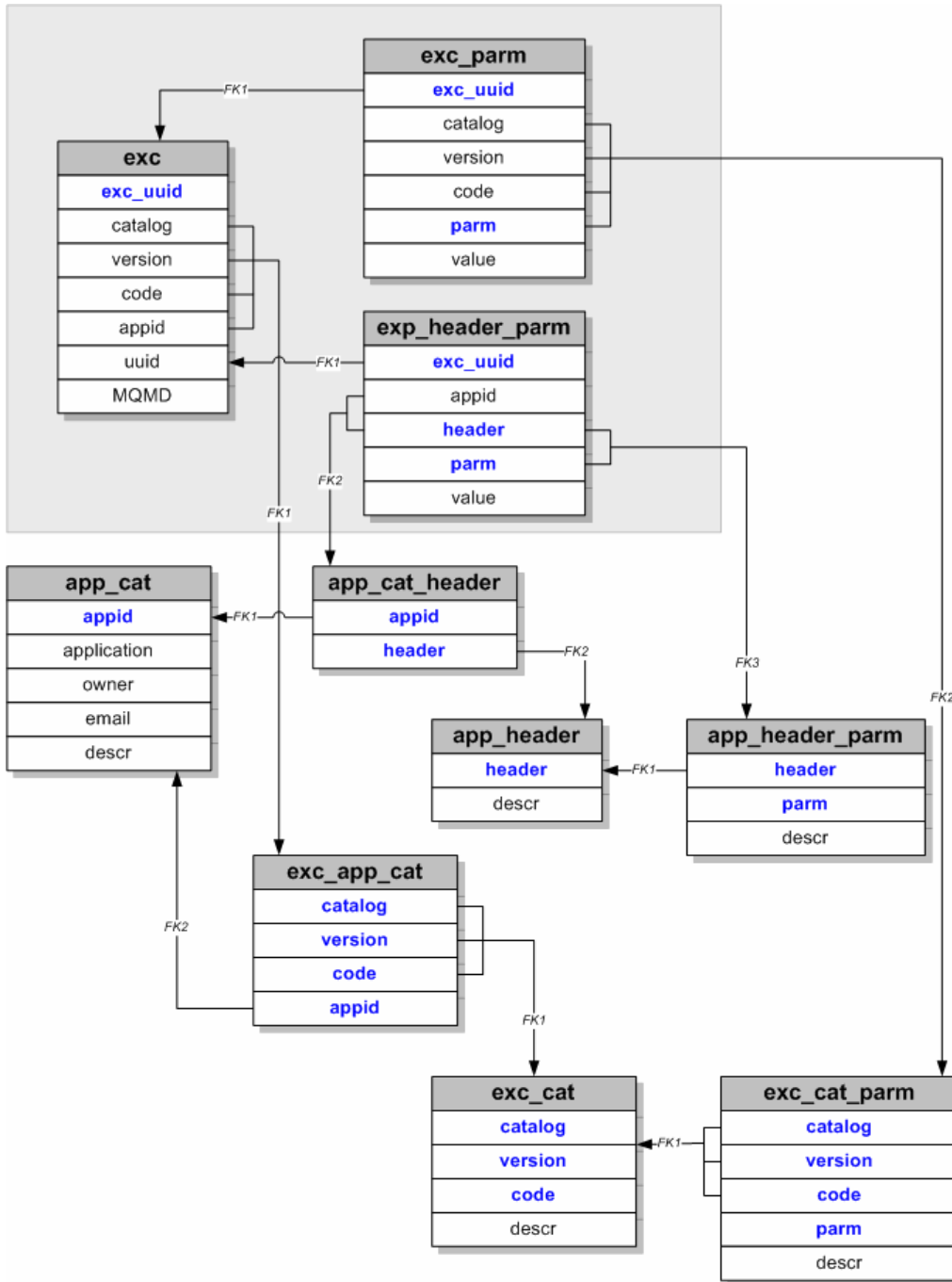


Figure 6 - Exception Database Schema



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.

<b>exc_cat – Exception Catalog</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
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
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		catalog, version, code
UK		catalog, version, name

<b>exc_cat_parm – Exception Catalog Parameters</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
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
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		catalog, version, code, parm
FK1	EXC_CAT	catalog, version, code

<b>exc_catagory – Exception Catagory</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
catagory	Char(2)	Excepcion Catagory 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 Catagory
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		catagory

<b>app_cat – Application Catalog</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
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 Applicaion
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		appid

<b>app_header – Application Catalog Header</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
header	Varchar(30)	Application Header
descr	Varchar(512)	Description of the Applicaion Header
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		header

<b>app_header_parm – Application Catalog Header Parameter</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
header	Varchar(30)	Application Header
parm	Varchar(30)	Application Header Parameter
descr	Varchar(512)	Description of the Applicaion Header Parameter
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		header, parm
FK1	APP_HEADER	header

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

<b>exc_app_cat – Association of the Exception for the Application</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
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
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		catalog, version, code, appid
FK1	EXC_CAT	catalog, version, code
FK2	APP_CAT	appid

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

<b>exc – Exception instance generated</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
<b>exc_uuid</b>	<b>Char(36)</b>	Universally Unique Identifier for the Exception instance
<b>timestamp</b>	<b>DateTime</b>	Timestamp when the exception was generated
<b>catalog</b>	<b>Varchar(30)</b>	Exception Catalog name for the project or organization
<b>version</b>	<b>Varchar(10)</b>	Exception Catalog version
<b>code</b>	<b>Char(5)</b>	Exception code: First 2 char for the category and the rest 3 are serial number
<b>priority</b>	<b>int</b>	Exception priority: 1 - Information, 2 - Warning, 3 - Error, 4 - Fatal
<b>appid</b>	<b>Varchar(30)</b>	Application Identifier
<b>hostname</b>	<b>Varchar(512)</b>	Hostname or IP address where Exception was generated
<b>function</b>	<b>Varchar(512)</b>	Function name where Exception was generated
<b>file</b>	<b>Varchar(1024)</b>	Program file name of Exception source
<b>line</b>	<b>Integer</b>	Line number of the program where Exception was generated
<b>server_cluster</b>	<b>Varchar(512)</b>	Cluster name for the server cluster. NULL for non-clustered servers, e.g. Cell for WebSphere Application Server
<b>server</b>	<b>Varchar(512)</b>	Server name for the server hosting the application. NULL for stand-alone applications, e.g. J2EE Application Server, WebSphere® Message Broker
<b>container</b>	<b>Varchar(512)</b>	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
<b>uuid</b>	<b>Varchar(32)</b>	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
<b>MQMD</b>	<b>Varchar(512)</b>	MQ Message Descriptor for the original message. This is base64 encoded
<b>MQMD_ver</b>	<b>Integer</b>	Version of the MQ Message Descriptor for the original message
<b>qmgr</b>	<b>Varchar(32)</b>	Queue Manager name for re-queuing of the input message
<b>queue</b>	<b>Varchar(32)</b>	Queue name for re-queuing of the input message
<b>dump_type</b>	<b>Varchar(512)</b>	The type of exception dump may be of the following flavors: <b>binary</b> – binary data dump <b>string</b> – any string type that may provide exception information <b>xml</b> – exception dumps represented in XML form excluding the XML declaration <b>exc</b> – embedded exception <b>java:&lt;Class&gt;</b> – serialized and Base64 encoded java class implementation of the interface DumpRenderer or an Exception class object.
<b>dump</b>	<b>blob</b>	Exception Dump
<b>data_type</b>	<b>Varchar(512)</b>	The type of exception dump may be of the following flavors: <b>binary</b> – binary data dump <b>string</b> – any string type that may provide exception information <b>xml</b> – exception dumps represented in XML form excluding the XML declaration <b>java:&lt;Class&gt;</b> – serialized and Base64 encoded java class implementation of the interface DataRenderer. <b>java:MQMessage</b> or <b>java:JMSMessage</b> – serialized and Base64 encoded for native MQ or JMS support.
<b>data</b>	<b>blob</b>	Source Data
<b>mod_data</b>	<b>blob</b>	Modified Data
<b>mod_usr</b>	<b>Varchar(256)</b>	Modifier User Identifier
<b>mod_time</b>	<b>DateTime</b>	Modification timestamp

<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		exc_uuid
FK1	EXC_APP_CAT	catalog, version, code, appid

<b>exc_parm – Exception instance parameters</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
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
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		exc_uuid, parm
FK1	EXC	exc_uuid
FK2	EXC_CAT_PARM	Catalog, version, code, parm

<b>exc_header_parm – Application Headers for the Exception instance generated</b>		
<i>Column</i>	<i>Type</i>	<i>Description</i>
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
<i>Key</i>	<i>Reference</i>	<i>Columns</i>
PK		exc_uuid, parm
FK1	EXC	exc_uuid
FK2	EXC_CAT_PARM	Catalog, version, code, parm

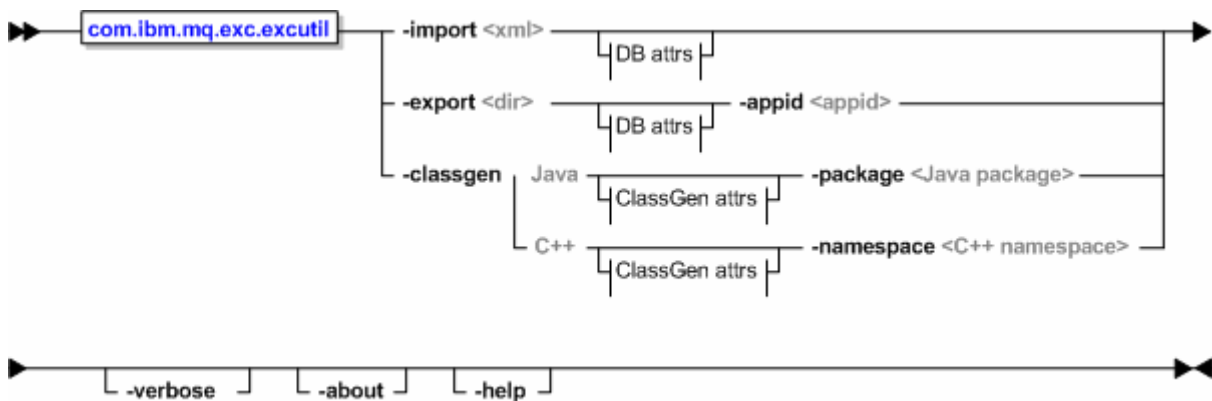
## 4. Exception Handler Runtime and Utilities:

### 4.1. 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:



#### DB attrs :

```

  -drv <JDBC driver> -url <URL> -usr <user id> -pwd <password>
  
```

#### ClassGen attrs :

```

  -classdir <dir> -exccat <xml>
  
```

**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** Application Identifier for the application whose related exceptions are to be exported. This is an optional parameter only relevant 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.

<sup>1</sup>Examples<sup>2</sup>:

- 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
```

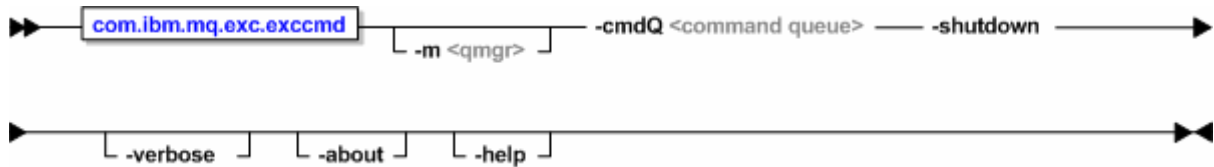
---

<sup>1</sup> Remember to set up the `EEH_PATH` environment variable to the EEH installation path.

<sup>2</sup> 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.

## 4.2. Exception Handler Command

A Java API package is provided to facilitate the building of Java-based applications to adhere to the standards



```

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 not explicit.
- about**  
Version and related information.

<sup>3</sup>Examples<sup>4</sup>:

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

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

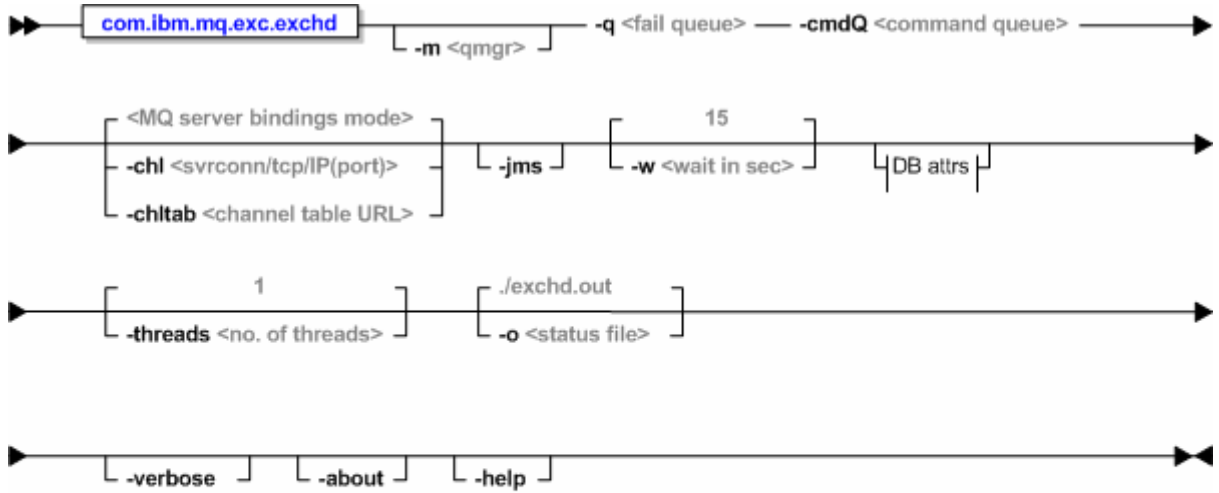
<sup>3</sup> Remember to set up the EEH\_PATH environment variable to the EEH installation path.

<sup>4</sup> Use the shutdown.bat or shutdown.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.



### 4.3. Exception Handler Daemon

A Java API package is provided to facilitate the building of Java-based applications to adhere to the standards



#### DB attrs :

```

  -drv <JDBC driver>
  -url <URL>
  -usr <user id>
  -pwd <password>
  
```

**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 notexplicit.  
**-about** Version and related information.

<sup>5</sup>Examples<sup>6</sup>:

- 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.COMD
```

---

<sup>5</sup> Remember to set up the EEH\_PATH environment variable to the EEH installation path.

<sup>6</sup> 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. 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<sup>7</sup> 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

-->
<exc:ExceptionCatalog name="ExcCat.MQST" version="1.0.0" appid="MQST"
  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 syntax 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 exists." />
    <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 occurred." />
    <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>
```

<sup>7</sup> 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<sup>8</sup> 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<sup>9</sup> 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');
```

---

<sup>8</sup> Sample location is <EEH\_PATH>/import.bat

<sup>9</sup> 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');
```

## 6. 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.

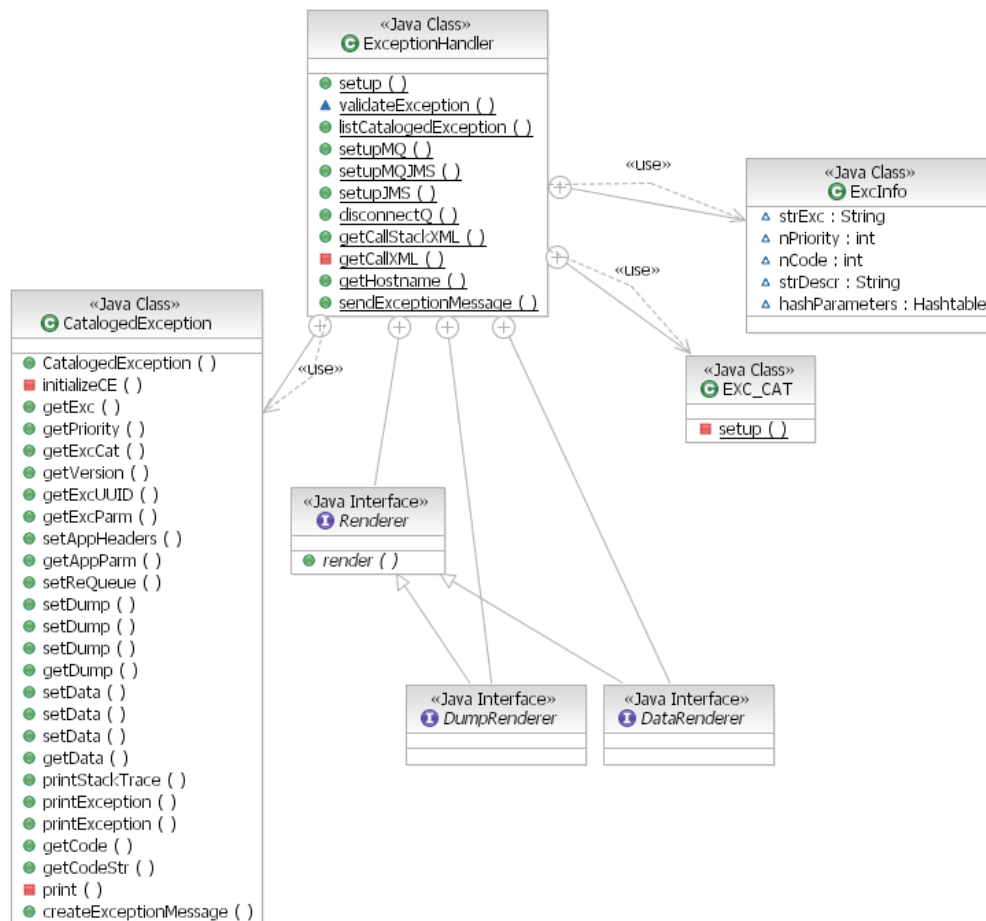


Figure 7 – Java API Class Diagram

---

## 6.1. 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<sup>10</sup>.

Following is a listing of the sample java class generation script<sup>11</sup> 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
```

---

<sup>10</sup> Please refer to the internal Exception Catalog in Appendix I:

<sup>11</sup> Sample location is <EEH\_PATH>/classgen.bat

The generated class provided in the samples<sup>12</sup> 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 strReferenceMissing 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);
    }

    .....
}
```

---

<sup>12</sup> Sample location is <EEH\_PATH>/samp/java/com/ibm/mq/exc/samp/ExcCat\_MQST.java



---

## 6.2. 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.

```
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);
}
```

### 6.3. 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;

```

---

## 6.4. 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) {
    CatalogedException 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);
```

---

## 6.5. 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.sendMessage(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();
    }
}
```

---

## 6.6. 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();
}
```

## 6.7. 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
```

## 7. 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.

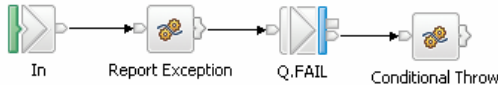


Figure 8 – ESQ\_Exception Subflow

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

## 7.1. 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.

### 7.1.1. 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;
```

### 7.1.2. 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);
    }
}
```



---

## **7.2. Generating Runtime Exceptions**

---

### **7.2.1. 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;
```

### 7.2.2. 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");

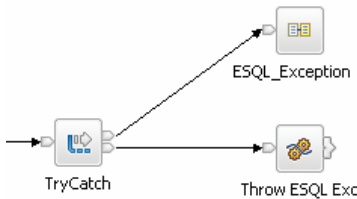
    try {
        Hashtable<String,String> hashErrorParms
            = new Hashtable<String,String>();
        hashErrorParms.put("InvalidXMLNS", "test");
        CatalogedException ceInvalidXMLNS
            = new CatalogedException(ExceptionHandler.PRI_FATAL,
                "EXC_XMLNS_VALIDATION", hashErrorParms, 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());
    }
}

```

## 7.3. Capturing and Reporting Runtime Exceptions

### 7.3.1. ESQL

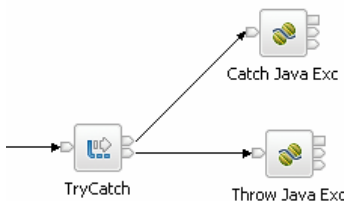


The exceptions caught in the message flow may be sent to the **ESQL\_Exception** sub-flow for capturing the exception as well as sending the formatted XML message to the exception queue.

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

Figure 9 – Catching Exceptions with the Subflow

### 7.3.2. 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 10 – 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.sendMessage(ce);

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

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

---

## 7.4. 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.

## **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 version="1.0" encoding="UTF-8"?>
<exc:ExceptionCatalog name="ExcCat.ExcH" version="1.0.0" appid="null"
  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 occurred."/>
    <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 exists."/>
    <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>
```



## Bibliography:

1. WMQ Support pack md08 – WebSphere MQ Network Design Notation  
[http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24006700&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24006700&loc=en_US&cs=utf-8&lang=en)
2. OMG 03-08-02 – UML 2.0 Superstructure specification
3. The Unified Modeling Language Reference Manual by James Rumbaugh, Ivar Jacobson and Grady Booch
4. WMQ Manuals:

Multiplatforms			
Title	Order number	PDF/size	Date
<a href="#">Monitoring WebSphere MQ</a>	SC34-6593-00	2.79MB	May 2005
<a href="#">WebSphere MQ Application Programming Guide</a>	SC34-6595-01	6.18MB	Mar 2006
<a href="#">WebSphere MQ Application Programming Reference</a>	SC34-6596-00	7.5MB	May 2005
<a href="#">WebSphere MQ Bibliography and Glossary</a>	SC34-6603-00	0.96MB	May 2005
<a href="#">WebSphere MQ Clients</a>	GC34-6590-01	1.82MB	June 2005
<a href="#">WebSphere MQ Constants</a>	SC34-6607-00	2.63MB	May 2005
<a href="#">WebSphere MQ Intercommunication</a>	SC34-6587-00	6.98MB	May 2005
<a href="#">WebSphere MQ Messages</a>	GC34-6601-00	3.87MB	May 2005
<a href="#">WebSphere MQ Migration Information</a>	SC34-6604-01	0.44MB	Oct. 2005
<a href="#">WebSphere MQ Programmable Command Formats and Administration Interface</a>	SC34-6598-01	0.44MB	Sept 2006
<a href="#">WebSphere MQ Publish/Subscribe User's Guide</a>	SC34-6606-00	1.63MB	May 2005
<a href="#">WebSphere MQ Queue Managers Clusters</a>	SC34-6589-00	2.04MB	May 2005
<a href="#">WebSphere MQ Security</a>	SC34-6588-01	2.00MB	Oct. 2005
<a href="#">WebSphere MQ Script (MQSC) Command Reference</a>	SC34-6597-01	2.00MB	Sept 2006
<a href="#">WebSphere MQ System Administration Guide</a>	SC34-6584-01	6.55MB	Mar 2006
<a href="#">WebSphere MQ Transport for SOAP</a>	SC34-6651-00	0.9MB	May 2005
<a href="#">WebSphere MQ Using C++</a>	SC34-6592-00	1.58MB	May 2005
<a href="#">WebSphere MQ Using Java</a>	SC34-6591-01	4.84MB	Mar 2006
<a href="#">WebSphere MQ Using .Net</a>	GC34-6605-01	0.8MB	Mar 2006

5. WMQ Support Pack ms03 – Save Queue Manager
6. WMQ Support Pack ms0e – MQ Administration Wrapper
7. IBM Alphaworks performance harness - <http://www.alphaworks.ibm.com/tech/perfharness>
8. IBM ESB Overview  
<http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esbbenefits>

9. Improve the performance of your WebSphere Business Integration Message Broker V5 message flow

[http://www.ibm.com/developerworks/websphere/library/techarticles/0406\\_dunn/0406\\_dunn.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0406_dunn/0406_dunn.html)

10. Books:

- WebSphere Message Broker Basics  
<http://www.redbooks.ibm.com/redbooks/pdfs/sq247137.pdf>
- Managing WebSphere Message Broker Resources in a Production Environment  
<http://www.redbooks.ibm.com/redbooks/pdfs/sq247283.pdf>
- WebSphere MQ V6 Fundamentals on this link in the Red Books section  
<http://www-306.ibm.com/software/integration/wmq/library/>
- WebSphere MQ Queue Manager Clusters  
<http://www.elink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC34658900>

11. Classes:

- WebSphere Message Broker V6 Application Programmer Workshop  
[http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course\\_description&courseCode=MQ661](http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=MQ661)