Enterprise Test Robot

Arunava (Ron) Majumdar
Sr. IT Specialist
Focused Technology Practice
arunava@us.ibm.com
IBM

WebSphere. software

Contents

Table of Figures

Modification History

Date	Version	Author	Description
07/03/2014	1.0.0	Arunava Majumdar	Final release for Support Pac it01

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 date 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:

While leading a team for a customer project and moving into the testing phase with Message Broker flows, we soon realized we did not have an adequate tool in the market that would be able to automate flow testing. The tool was not only required to send messages to one or more queues from a file system or queue but also to retrieve messages and validate them against know results. All the requirements went into the building of the asset and we could deliver automated test for unit, system, performance and quality assurance environments. I would like to acknowledge all the team members who worked with me at the time to make this a success. The tool continues to evolve as new requirements are accepted as features.

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/).
This product includes software developed by the iText Project (http://www.lowagie.com/iText/).
This product includes software developed by the Trilead SSH (http://sourceforge.net/projects/orion-ssh2/files/)

Scope of the Document:

The scope of the document is limited to the automation of testing that may be achieved through the Enterprise Test Robot tool. The various features of the tool along with examples and scenario are presented in this paper. The document would outline each of the implemented parameters in the tool.

The ETR tool is used for non-invasive and repeatable test automation. It is not within the scope to monitor the system or any other specific products. There are numerous monitoring products available and the intent of the tool is not to replicate the functionality. It also does not capture data from the screen and replay it. There are several tools in the market to record screen actions and replay them.

ETR is geared towards and optimized for middleware testing, especially with queues. Options for transformation of data, regulation of the feed into the application and validation options to compare and decide if the test was successful or was a failure are all within the scope of the document and part of the ETR tooling.

To my wife Sonia Mahajan
for her support during the creation and publication of this asset

1. Introduction to Automated Testing:

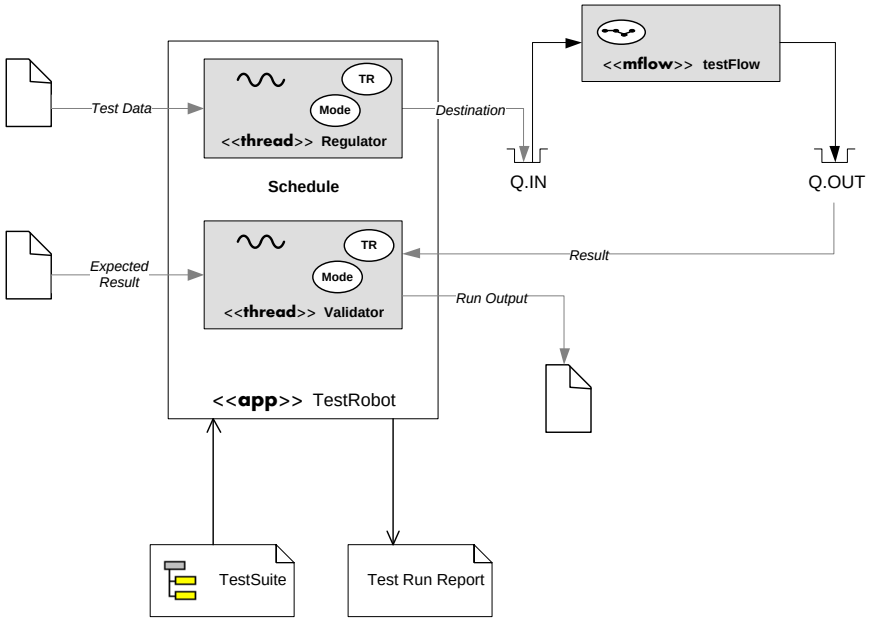


Figure 1 - Introduction to Test Automation

The first step to the automation of testing is to understand how Test Data and Expected Results may be saved so that the results of the test can be repeatable. The data saved from the system often contains elements that change and cannot be injected into the system directly but with some intervention. This intervention may be of several types – date modifications, counters that must be incremented, UUID generation, etc. On the other hand the saved canned data suffers from the same issues when being compared against. The tool provides transformation capabilities to deal with these situations and several modes for the validation of data – Search, Ignore, Sequence, etc. The regulation of data may also take place under several modes – Rate Sensitive, Coordinated, etc. Identifying how to send the data and how to validate it against known data is key to moving on to the next step in automation of the process.

The diagram above shows a generic testing scenario where the Test Data is picked up by the Regulator from a file and send to a Queue that provides the input to the application being tested. The application in this case is a Message Flow that reads the data transforms it and puts it into another Queue. The Validator then picks up the data from the output Queue, validates against the known Expected Results saved and makes a decision whether the test passed or failed. The data is also saved to the Run Output location so that it may be analyzed when a test fails. The configuration of the Regulator and Validator and any transformations required are provided in a simple XML format – the Test Suite. For each Test Run a report is generated for convenience.

The Test Suite is a collection of Test Cases and any Load and Unload of data for the Test Run. Each Test Case may have multiple Regulators and Validators. Test Data, Results, Expected Results and Run Output are all considered as Data Sources.

2. Testing Phases:

Most organizations have their own definitions of testing phases and often it is unclear in most books on the subject the exact definition of the different phases. This section attempts to define the phases of testing pertaining to how Enterprise Test Robot is configured. The same tests may be performed in multiple environments and should not change the definition of the test itself. Often certain tests are not performed in certain environments due to constraints on the environment, e.g. running performance test in the development environment with limited resources allocated may bring it to its knees. On the other hand, the QA environment receiving a parallel feed from the PROD environment at 30 msg/sec will not be able to check each and every message data. We certainly hope it went through rigorous testing of functionality in prior environments.

2.1. Unit Testing

- Testing for the individual applications in a controlled manner
- All the functionality for the application must be individually tested with specific Test Cases
- All exception conditions must be tested and validated if the exception reporting is proper
- Regression Tested for any changes to the application
- Several cycles of testing/bug fixing may be required – Test Runs
- All Test Cases run with canned data

2.2. System Testing

- Testing for the a set of applications in a controlled manner
- Integration points must be validated
- Validations are based on system-wide results
- Regression Tested for any changes to any application after all the Unit Test cases are validated
- Maximum volume or message capacity may be tested
- Several cycles of testing/bug fixing may be required – Test Runs
- All Test Cases run with saved production data

2.3. Performance Testing

- Testing the performance of the application individually or at a system level
- Data sent to the application at various rates
- Parallel Run or simulation of the production load curve at various levels
- The maximum rate capacity may be tested
- The environment where the test is run must be comparable to the production environment
- All Test Cases run with saved production data or a live production feed

2.4. User Acceptance Testing

- Especially with user-interface based when multiple users pound the system with various sorts of data as well as testing of concurrent system activities
- Parallel Run or simulation of the production load curve at various levels
- The environment where the test is run must be able to withstand the load
- All Test Cases run with saved production data or a live production feed

3. Getting Started with Test Automation

Before we can get started with ETR, first of all we need to find out what we would like to test and how we can test the outcome of the application. Many organizations already have documented test cases that list the conditions for the test and what the outcome would look like. When we start thinking about automating the test case, we need to first find some test data that we can send to the application so that it exercises the conditions for the test.

Consider an application with a simple if statement that checks for a string value of a field (**type**) and based on that value produces two different outputs. Let us assume that if the value is true it produces a fixed length string and if the value is false produces an XML. The incoming data is comma (,) delimited.

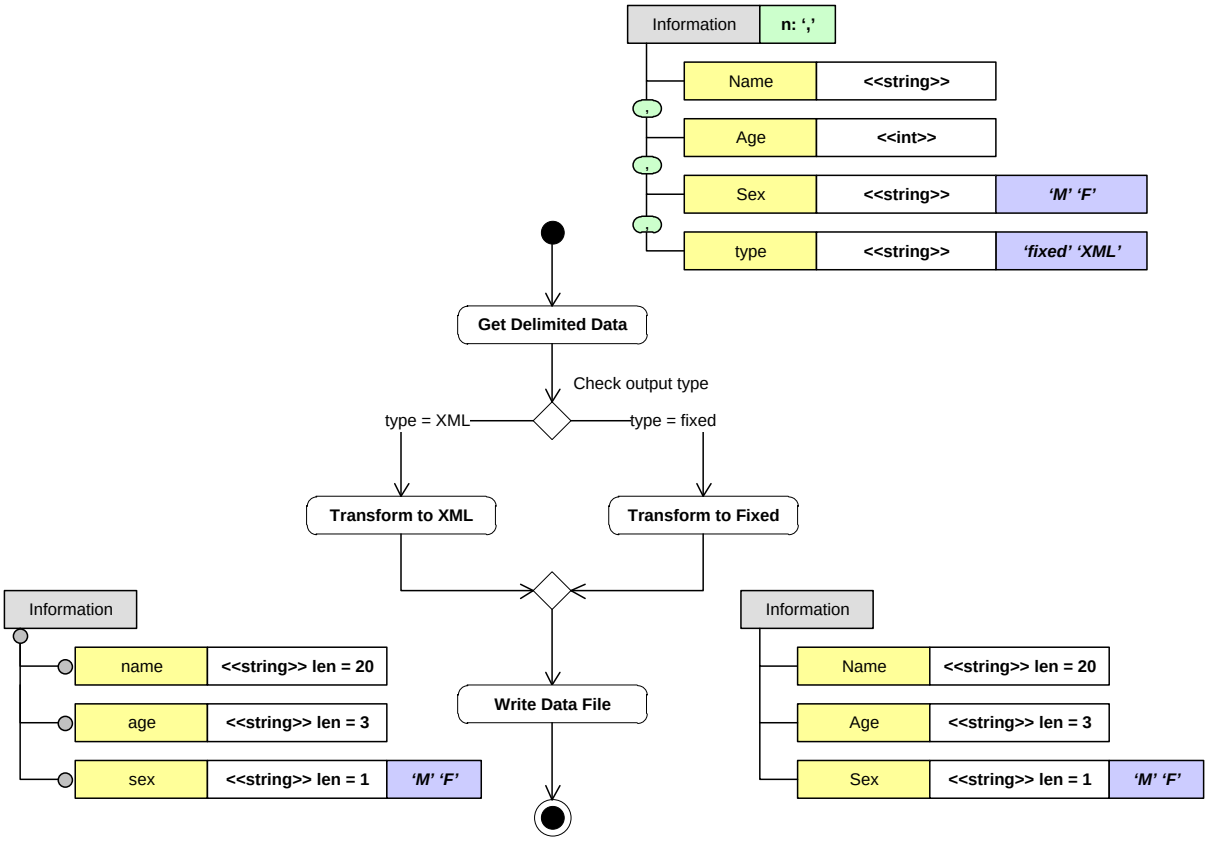


Figure 2 – Simple Transformation Application (Personal Information Transformer)

For the simple application (Personal Information Transformer) described, we have two conditions to be tested. Before proceeding further let us assign an application id to the application – PIT. This would help us refer to any artifact related to this application.

Case #	Exc Code	Condition	Ref Function	Assumption	Test Data & Script	Expected Result
PIT.01		Delimited personal information is transformed to Fixed Length file [type = 'fixed']			PIT.01.dat	PIT.01.fixed
PIT.02		Delimited personal information is transformed to XML file [type = 'XML]			PIT.02.dat	PIT.02.xml

Notice that in the table above we have mentioned the names of the files providing input test data and output expected data. Now let's look into the contents of these files.

Condition PIT.01:

PIT.01.dat

```
John Doe,40,M,fixed
```

PIT.01.fixed

```
John Doe            040M
```

Thus, to test condition PIT.01 it is sufficient to send the file PIT.01.dat and if the output result **exactly matches** with the expected output file prepared PIT.01.fixed, then the test case **passes**. Else it **fails**.

This is the simplest form of test since the output is always predictable and should exactly match the expected results file for the test to pass. We will look into scenarios in later chapters where data contains date and time and other fields that are validated in the application and the test cannot be performed unless the date is current and the test result cannot be determined to have passed or failed unless we ignore certain sections of the data.

Condition PIT.02:

PIT.02.dat

```
Jane Doe,20,F,XML
```

PIT.02.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Information name="Jane Doe" age="20" sex="F"/>
```

Similarly, to test condition PIT.02 it is sufficient to send the file PIT.02.dat and if the output result **exactly matches** with the expected output file prepared PIT.02.xml, then the test case **passes**. Else it **fails**.

We will see in later chapters how this validation may be enhanced in the next version.

Now that we have Test Data, Expected Results and a definitive validation method that may in theory be repeated, let us see how the Enterprise Test Robot can help us automate this test.

The application is listening for the arrival of new files in a specific directory and outputs the resultant files to another configured directory. ETR needs to send the test data file from one location to another location, essentially copying the file. On the validation side, ETR needs to pick up the result file form one location, compare it with the saved expected results file from another location and determine if they both match byte-by-byte (exact match) and decide whether the test passed or failed.

In ETR, the component that sends test data is called **Regulator** and the component that validates the data and makes the determination of the passing or the failure of the test is called **Validator**.

Here is how we can set the Regulator:

```
<Regulator>
    <TestData><File path="data/PIT.01/PIT.01.dat" format="raw"/></TestData>
    <Destination><File path="in/PIT" format="raw"/></Destination>
</Regulator>
```

Both **TestData** and **Destination** are ***Abstract*** **Data Sources**. The implementations include **FileSource** and **QueueSource**. Hence the Test Data may be sourced from either file or queue and the destination may be either file or queue. As Data Source implementations are added all combinations are possible, making it flexible and powerful.

Similarly the Validator is set as follows:

```
<Validator>
    <Latency hr="0" min="0" sec="5"/>
    <Results><File path="out/PIT" format="raw"/></Results>
    <ExpectedResults><File path="results/PIT.01/PIT.01.fixed" format="raw"/></ExpectedResults>
    <RunOutput><File path="" format="raw"/></RunOutput>
</Validator>
```

Latency provides a way to delay the validation from starting immediately and, in this case, gives the PIT application 5 seconds to process the data. **Results** and **ExpectedResults** are also Abstract Data Sources. The **RunOutput** location stores the output data information for inspection after the test is run.

The simplest **Regulator** mode is "**unregulated**" and the simplest **Validator** mode is "**exact match**". To automate these simple Test Cases, that is all we need. The ETR is a Java 1.5 application that runs like a command and the only mandatory parameter is **–config** pointing to the configuration XML file for the Test Suite. The file extension does not matter in the case of the command line mode. However, it must be **.testsuite** for use in the eclipse plug-in. It is good practice to have a consistent **.testsuite** extension.

Let us now look at the configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<mqt:TestSuite xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="urn:com.ibm.mq.test TestConfig.xsd"
               xmlns:mqt="urn:com.ibm.mq.test"

               author="Arunava Majumdar"
               version="1.0.0" project="ETR.Test"
               appid="PIT" application="Personal Information Translator"
               appOwner="Sonia Mahajan"
               testType="unit"
               appTestPath=""
               appTestRunPath="runs">

       <TestCase id="PIT.01" descr="Exact Match - File Regulator">
               <Condition>
                       Delimited personal information is transformed to Fixed Length file
                       [type = 'fixed']
               </Condition>
               <Regulator>
                       <TestData><File path="data/PIT.01/PIT.01.dat" format="raw"/></TestData>
                       <Destination><File path="in/PIT" format="raw"/></Destination>
               </Regulator>
               <Validator>
                       <Latency hr="0" min="0" sec="5"/>
                       <Results><File path="out/PIT" format="raw"/></Results>
                       <ExpectedResults>
                               <File path="results/PIT.01/PIT.01.fixed" format="raw"/>
                       </ExpectedResults>
                       <RunOutput><File path="" format="raw"/></RunOutput>
               </Validator>
       </TestCase>

       <TestCase id="PIT.02" descr="Exact Match - File Regulator">
               <Condition>
                       Delimited personal information is transformed to XML file
                       [type = 'XML']
               </Condition>
               <Regulator>
                       <TestData><File path="data/PIT.02/PIT.02.dat" format="raw"/></TestData>
                       <Destination><File path="in/PIT" format="raw"/></Destination>
               </Regulator>
               <Validator>
                       <Latency hr="0" min="0" sec="5"/>
                       <Results><File path="out/PIT" format="raw"/></Results>
                       <ExpectedResults>
                               <File path="results/PIT.02/PIT.02.xml" format="raw"/>
                       </ExpectedResults>
                       <RunOutput><File path="" format="raw"/></RunOutput>
               </Validator>
       </TestCase>
</mqt:TestSuite>
```

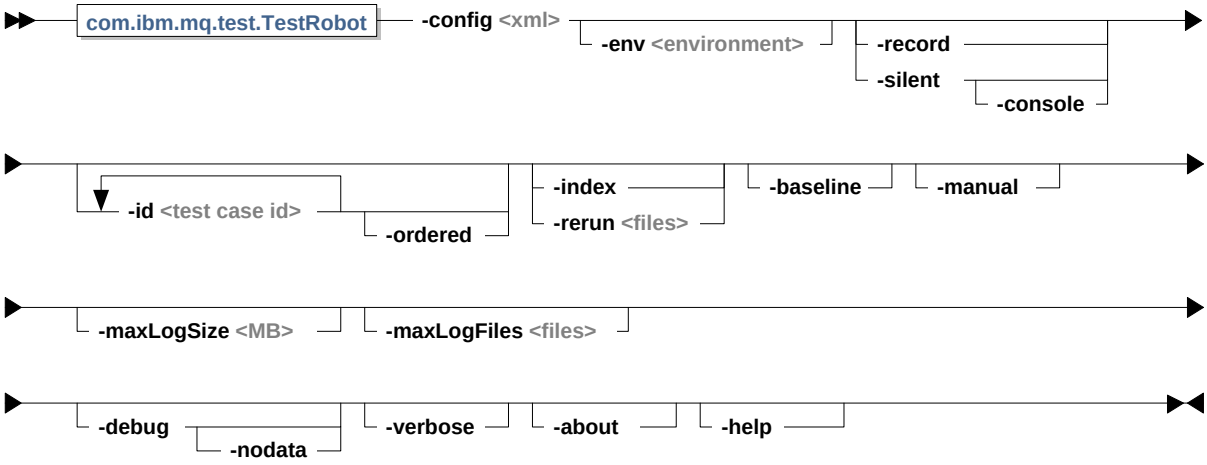
The environment variable **ETR_PATH** must be set to the installation directory for ETR. The path should only have the universal file separator '/' and must not end with a '/'. A batch file is provided to facilitate the loading of the dependent jars[1].

```
ETR_PATH=C:/IBM/Assets/ETR
call bin/win/TestRobot.bat -config samp/PIT/unit/PIT_UT.testsuite
```

[1] **Additional Jar files may be required for connecting databases, etc.**

4. ETR Command Syntax

The Enterprise Test Robot provides a simple command line interface to launch as well as control the flow of the Test Cases in the Test Suite. The Test Suite is represented as the configuration XML. Multiple environment setups may be provided in the Test Suite and controlled from the command line. This is useful for maintaining the same XML file with separate connection parameters for different environments or different users and controlled from the command. **Record** and **Silent** options let the user save the Test Run information to a profiles file. The **id** lets the user select specific Test Cases only and optionally reorder them. For nested directories, the Test Data is indexed relative to the location for portability and reused for performance. However, the directory has to be **re-indexed** if the data set is changed. So of the data in the data set may also be **rerun** for large data sets that fail. When the application is tested and the results are verified the stream can be reversed in the **baseline** mode and the **Results** may be saved as the **Test Data**. The Test Run may also be in **manual** mode that stops after every Test Case for acknowledgement and is useful while setting up the Test Cases.



```
com.ibm.mq.test.TestRobot
        -config <Test Configuration XML>
        [-env <Environment>]
        [[-id <Test Case Id>]n [-ordered]]
        [-baseline] [-manual]
        [(-record) | (-silent [-console])]
        [(-index)|(-rerun <Rerun Files>)]
        [-maxLogSize <size in MB>]
        [-maxLogFiles <no of cycled files>]
        [-debug [-nodata]]
        [-verbose] [-about] [-help]
```

-config
The XML file for the test suite definitions conforming to the schema TestConfig.xsd.

-env
(Optional) The Environment where the Test is being run.

-id
(Optional) Test Case Id defined in the Test Suite to be run.
This can be repeated as many cases are provided. The test cases are run in the order are presented on the command line.

-baseline
(Optional) Saving test results for baselining.

-manual
(Optional) Manual Mode for testing. Haults before each test case for user input. Type 'exit' for stop running the remaining test cases.

-record
(Optional) Saving the input data in a recording file.
Cannot be used with the **-silent** option.

-silent
(Optional) Playing back the input data from a recording file.
Cannot be used with the **-record** or **-manual** options.
Turns off writing to console and starts saving information in TestRobot.system.out file.

-console
(Optional) Turn writing to console on in the silent mode.

-index
(Optional) Turn on indexing for all files. If index exists, the index is recreated.

-rerun
(Optional) Skips ahead to the last file processed in the index and reruns the number of files specified in the last run. Rerun files = 0 implies that the testing resumes at the position of failure of the last run as saved in the index.
Cannot be used with the **-index** option.

-maxLogSize
(Optional) The maximum size in MB each log can grow to. A new log is created when the limit is reached. The log is not cycled unless set with **-maxLogFiles** parameter.
Default value is 10 MB.

-maxLogFiles
(Optional) Active logs are cycled if the parameter is set to the number of files to be maintained. Files older than the number of active files are deleted.
Default value is -1 (Log cycling is turned off).

-debug
(Optional) This sets the debug trace for the tool. This provides detailed level tracing for the tool that may be required for troubleshooting the tool itself.

-nodata
(Optional) Only applicable for -debug mode to turn off any data reporting in the log if not required to reduce the size of the log.

-verbose
(Optional) Parameter only to be used for debugging if errors are not explicit.

-about
(Optional) Version and related information.

5. ETR Field Reference

The Enterprise Test Robot is based on a simple XML configuration for automating your tests. This chapter provides information on the over-all structure and the configuration details of each of the fields.

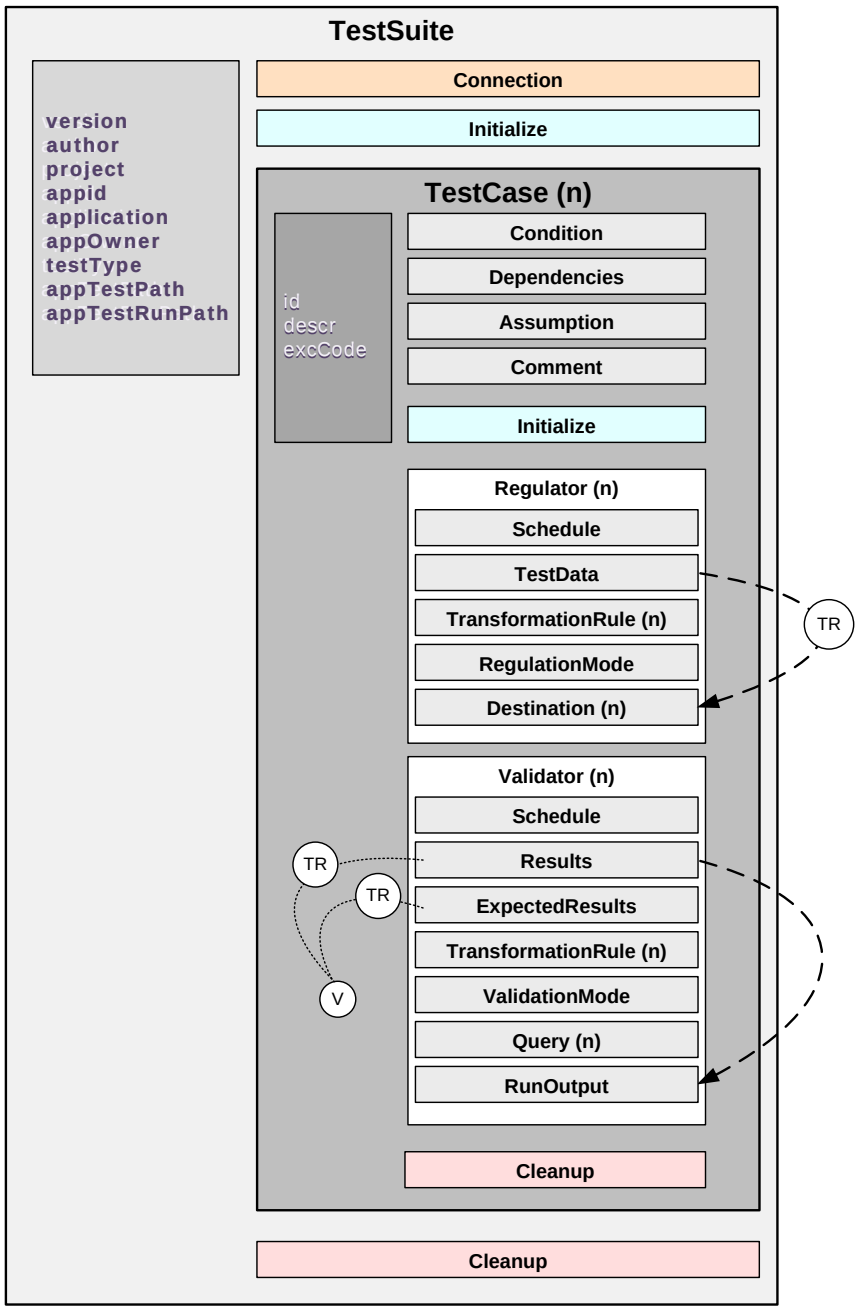


Figure 3 – ETR Configuration Overview

The diagram shows the basic structure of the Test Suite Configuration. Certain fields are provided for documentation purposes only. The Test Suite attributes version, author, project, etc. are provided for information

related to the Test Suite. In general a Test Suite is created for every application and testing phase. In case of large applications, a Test Suite may be based on each feature. As emphasized before in earlier chapters, having an application identifier (appid) to represent the application is good practice and all the application artifacts can now use the same appid. Connections are defined at the global space and may be used by any Test case. The Initialization and Cleanup routines may be defined both globally and within the Test Case scope.

Multiple Test Cases may be defined in the Test Suite. Test Cases are synchronized in the Test Suite, i.e. the next test case to be executed always waits for the previous test case to be completed. This gives control over the atomicity of the test. Some test cases may have dependent test cases. This means that the dependent test case is only run on the successful completion of its dependencies. Multiple dependencies may be defined.

Conditions, Assumptions and Comments are for documentation only.

The core functionality of the test is performed by the Regulators and the Validators. Multiple Regulators and/or Validators may be assigned to a Test Case. Regulators send Test Data to a Destination and Validators introspect the Results of the test and decides on the verdict of passed or failed.

Both Regulators and Validators can be scheduled for a specific Test. A regulator may be started only at the top of the hour to test a condition that the application gathers information of the last hour and in reality will be triggered from Unix cron job. A validator may be started with a latency of 10 seconds for the application to process some very complex transformations and produce the result in 10 seconds service level agreement (SLA).

Both Regulators and Validators can perform multiple transformations on the data. In case of the regulator the test may want to replace a timestamp field to the current timestamp. The validator, on the other hand, may chose to replace the timestamp from the result to X's and match the result with the existing Expected Results markes with X's for the rest of the fields.

Both Regulators and Validators have their individual modes of operation; the simplest being "Unregulated" and "Exact Match" respectively when their modes are not mentioned.

Additionally, queries may be run from the Validator, in conjunction with Results or by themselves, to validate records from the database and compare them to the data saved in files. Please refer to the Queries section for more details.

Validator output is always saved at the Run Output location. Only File Data Source is supported in this release.

5.1. Test Suite

At the top level for the configuration of the Enterprise Test Robot is the Test Suite definition. The Test Suite encapsulates all the Test Cases, Connection parameters used by the tests defined and an Initialization and Cleanup section for the setting up the test environment and cleaning it up.

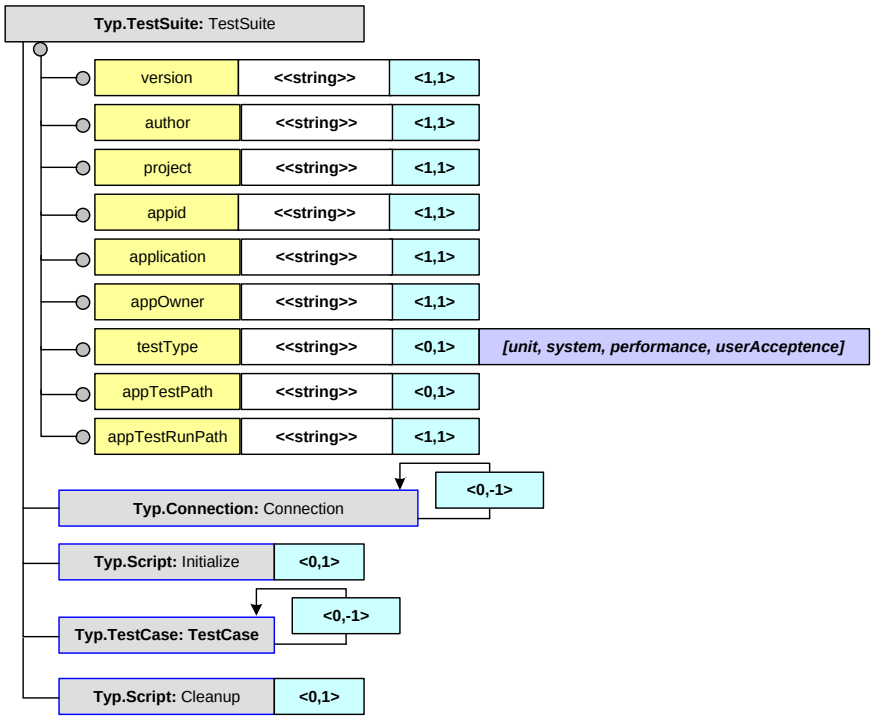


Figure 4 – Test Suite Definition

Field	*Description*
version	The version of Test Suite.
author	Author of the Test Suite.
project	Project name, if the test is being performed by a capitalized project.
appid	Application Identifier for the application. This id is used to identify all related artifacts for the application.
application	Full name of the application.
appOwner	Owner of the Application.
testType	Type of testing – '***unit***', '***system***', '***performance***', '***userAcceptance***'
appTestPath	The path where the test artifacts are kept. The relative path is set to the location of the XML Configuration file. Absolute paths may be set. Multiple environment variables may be set in the string for path abstraction and are resolved at runtime. All paths in the resolved path must follow the file separator '/' and should not end with the '/' indicator.
appTestRunPath	Similar to appTestPath but points to the directory where the test run files are stored.
Connection	Connection parameters are declared globally in the Test Suite. Supports parameters for multiple environments and directed to by the ***–env*** command line parameter.
Initialize	Test Suite initialization scripts.
TestCase	Test Case definitions for the Test Suite.
Cleanup	Test Suite cleanup scripts.

5.2. Connection

All Connections for the Test Suite is defined at the global level. Each Connection is identified by its key at the Test Suite scope and may be referred to by Data Sources, Queries and Scripts.

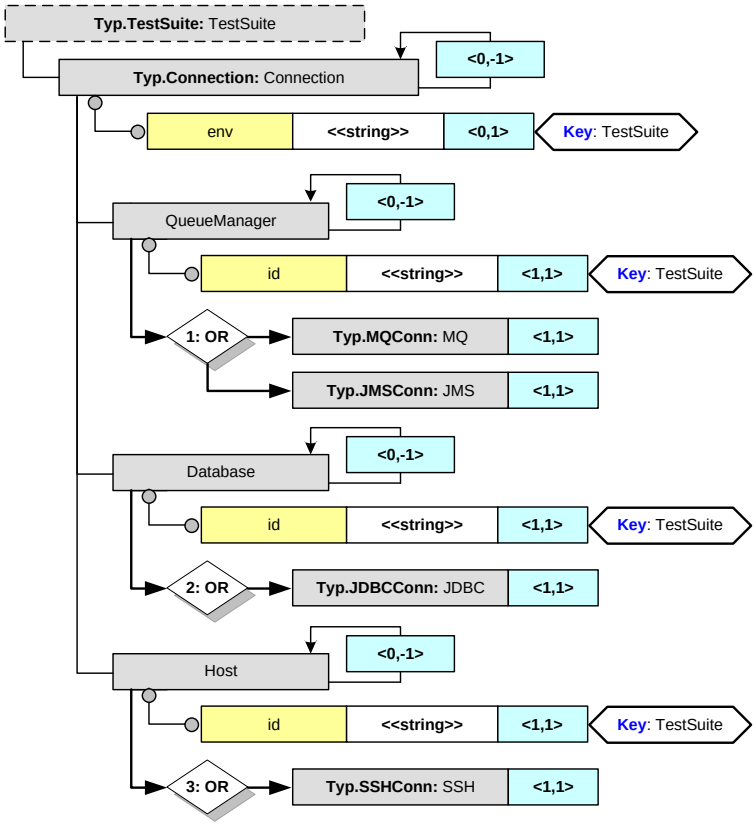


Figure 5 – Connection Parameters

Field	*Description*
env	Environment identifier. Multiple Connection parameters may be provided for different environments running the same tests. The ***–env*** command line parameter determines which Connection parameters are used for the test run. If no environment parameter is passed in the command line, the default Connection parameters are used where attribute is not specified. Exception is thrown is the environment is not defined.
QueueManager	Connection parameters for the Queue Manager.
id	Connection identifier for the Queue Manager.
MQ	MQ Connection parameters.
JMS	JMS Connection parameters. This feature is not implemented in this release.
Database	Connection parameters for the Database.
id	Connection identifier for the Database.
JDBC	JDBC Connection parameters.
Host	Connection parameters for connecting to a remote host.
id	Connection identifier for the remote host.
SSH	SSH Connection parameter.

5.2.1. MQ Connection

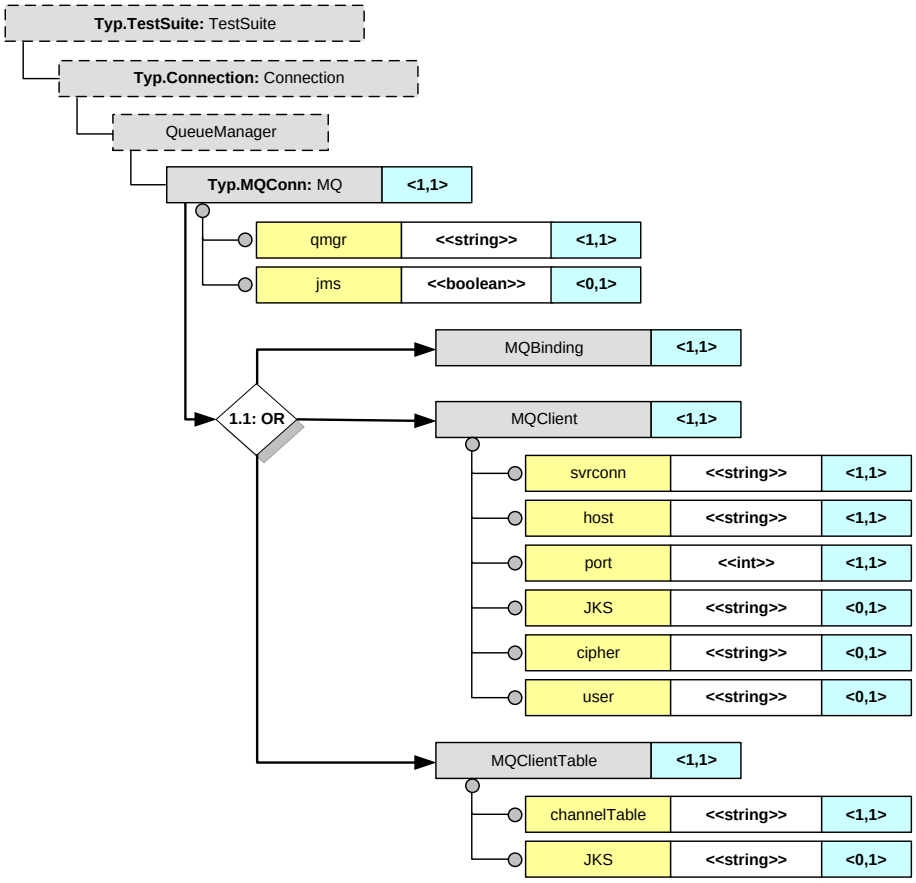


Figure 6 – MQ Connection parameters

Field		*Description*
qmgr		Queue Manager name.
jms		Java Messaging Service enabled (not supported in this release).
MQBinding		MQ Binding mode. This can only be used with a local Queue Manager.
MQClient		MQ Client connection mode. Local and remote Queue Managers with a Server Connection channel may be connected.
	svrconn	Queue Manager's Server Connection Channel name.
	host	Hostname, DNS name or IP address of the Queue Manager node.
	port	TCP/IP Listener port for the Queue Manager.
	JKS	Java Key Store to use with SSL Channels (not supported in this release).
	cipher	Cipher Specification to use with SSL Channels (not supported in this release).
	user	User Identifier for connecting to the Queue Manager. The password is not saved in the configuration file. It must be provided at runtime. The password may be saved in the user profile for the application by using the ***–record*** option. It is encrypted and saved in the profile and can only be retrieved at runtime from the ***–silent*** option.
MQClientTable		MQ Client Channel Table mode.
	channelTable	Client Channel Table for connecting to the Queue Manager.
	JKS	Java Key Store to use with SSL Channels (not supported in this release).

5.2.2. JDBC Connection

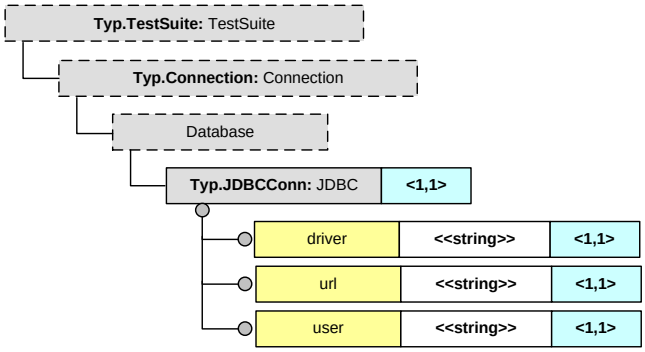


Figure 7 – JDBC Connection parameters

Field	*Description*
driver	JDBC Driver full class name.
url	Universal Resource Locator for connecting to the database.
user	User name to connect to the Database. The password is not saved in the configuration file. It must be provided at runtime. The password may be saved in the user profile for the application by using the ***–record*** option. It is encrypted and saved in the profile and can only be retrieved at runtime from the ***–silent*** option.

5.2.3. SSH Connection

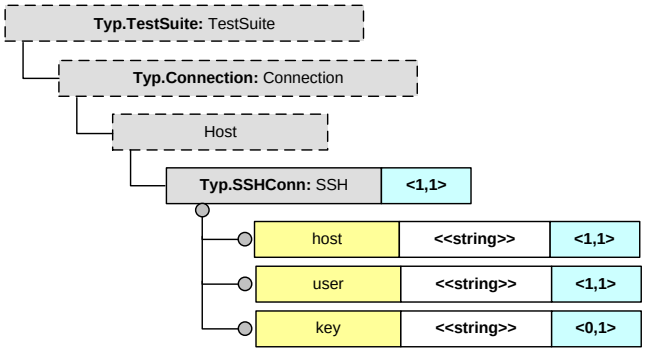


Figure 8 - SSH Connection parameters

Field	*Description*
host	JDBC Driver full class name.
user	User name to connect to the SSH Server. The password is not saved in the configuration file. It must be provided at runtime. The password may be saved in the user profile for the application by using the ***–record*** option. It is encrypted and saved in the profile and can only be retrieved at runtime from the ***–silent*** option.
key	Public key file to connect to the SSH Server.

5.3. Initialization and Cleanup

Both **Initialize** and **Cleanup** sections are of type Script. The sections may be configured both for the Test Suite and within the scope of the Test Case.

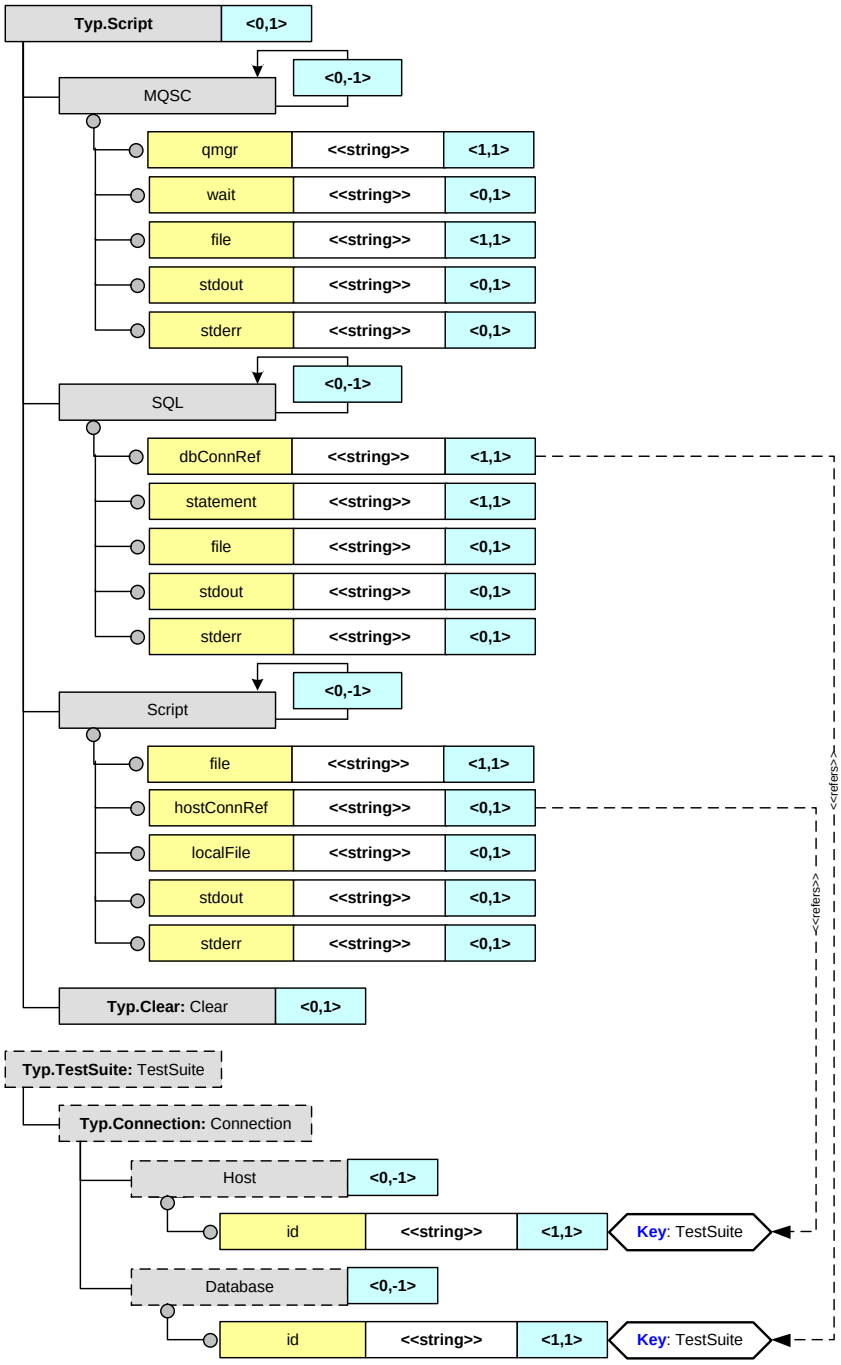


Figure 9 - Initiator and Janitor parameters

Field		Description
MQSC		IBM MQ Scripts. MQSC scripts can be run from the tool through the **runmqsc** command.
	qmgr	Queue Manager name.
	wait	Wait time in seconds when connecting to a remote queue manager through a locally bound one to execute commands remotely. The message channel pairs and the transmit queue must be set up for remote execution of commands to work. This is an encapsulation of the **–w** switch for **runmqsc**.
	file	The input file for the MQSC scripts.
	stdout	Standard output redirection to the file name specified.
	stderr	Standard error output redirection to the file name specified.
SQL		SQL Script to run Insert, Update, Delete statements in a database.
	dbConnRef	Database connection reference. The connection id for the database defined in the Connections section.
	statement	SQL statement to execute.
	file	File to provide input data for parameterized SQL Statements.
	stdout	Standard output redirection to the file name specified.
	stderr	Standard error output redirection to the file name specified.
Host		Running scripts on a remote host.
	file	File name to run on the remote host.
	hostConnRef	Host connection reference. The connection id for the host defined in the SSH Connections section.
	localFile	Local filename that can be transferred to the remote host if the file is not found at the file location on the host and then run on the server. This provides a convenient method to run the script on a remote host without having to FTP or SCP prior to running the test. (Not implemented in this release)
	stdout	Standard output redirection to the file name specified.
	stderr	Standard error output redirection to the file name specified.
Clear		Section to set what data must be cleared at the beginning or end of the test.

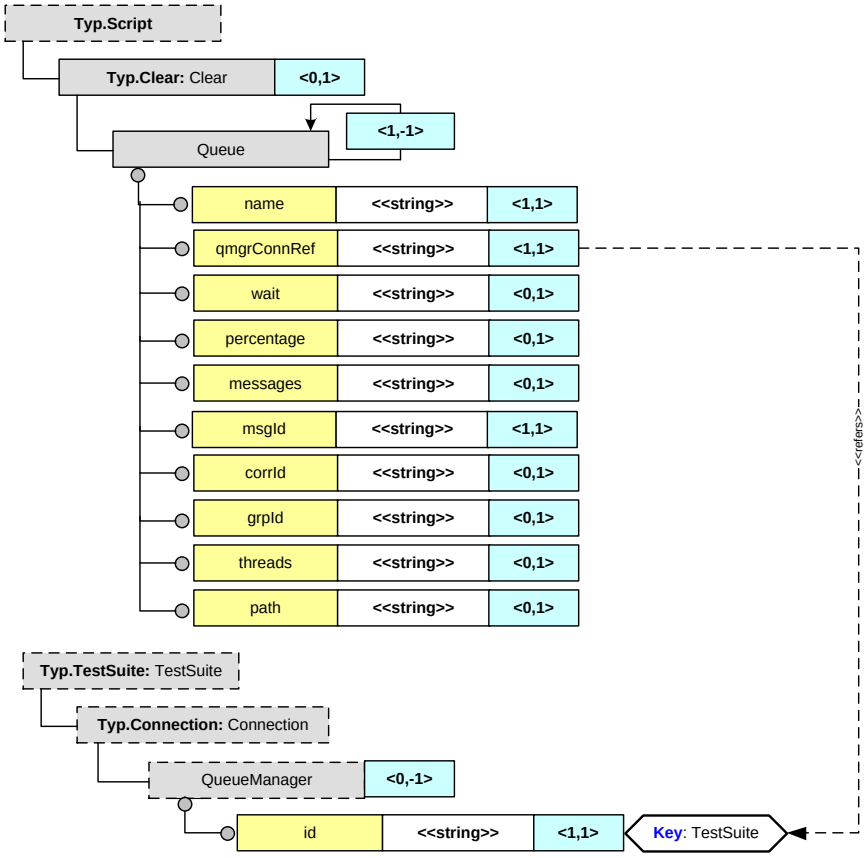


Figure 10 – Clear Queue parameters

Field	*Description*
Queue	Clear Queue parameters.
name	Name of the queue
qmgrConnRef	Queue Manager connection reference. The connection id for the queue manager defined in the Connections section.
wait	Wait in seconds for new message arrival on the queue.
percentage	The percentage of messages to be deleted from the queue. The percentage is statically calculated at the start of the application. Should not be used with messages.
messages	Number of messages to be deleted from the queue. Should not be used with percentage.
msgId	Selected messages with the Message Id can be cleared.
corrId	Selected messages with the Correlation Id can be cleared.
grpId	Selected messages with the Group Id can be cleared.
threads	Number of threads spawned to clear the queue. In general only 1 thread is needed. But in case of performance or capacity testing, multiple threads may reduce the time to cleanup for the subsequent test runs.
path	Path where the messages may be saved.

5.4. Test Case

Multiple Test Cases are defined in the Test Suite. Test Cases are synchronized and only starts when the previous Test Case has ended. The Test Cases are run in the same order as defined in the configuration file. Specific Test Cases may be run from the command line with the **–id** switch. This provides a simple method for running a subset of the defined Test Cases without changing the configuration or making a copy. Moreover, the sequence of running the Test Cases can be altered for a specific Test Run with the **–ordered** switch.

Test Cases may also be made dependent on other Test Cases where the Results of the prior test affect the latter, e.g. Test Case **TC1** inserts data into a database and Test Case **TC2** uses the data to produce Results to be validated. Rather than deleting the data after validating that the inserted data is correct and re-inserting, it may be used for the dependent Test Case. Multiple dependencies may be defined with Predecessors. The dependent Test Case is only run if all the Predecessors are successful.

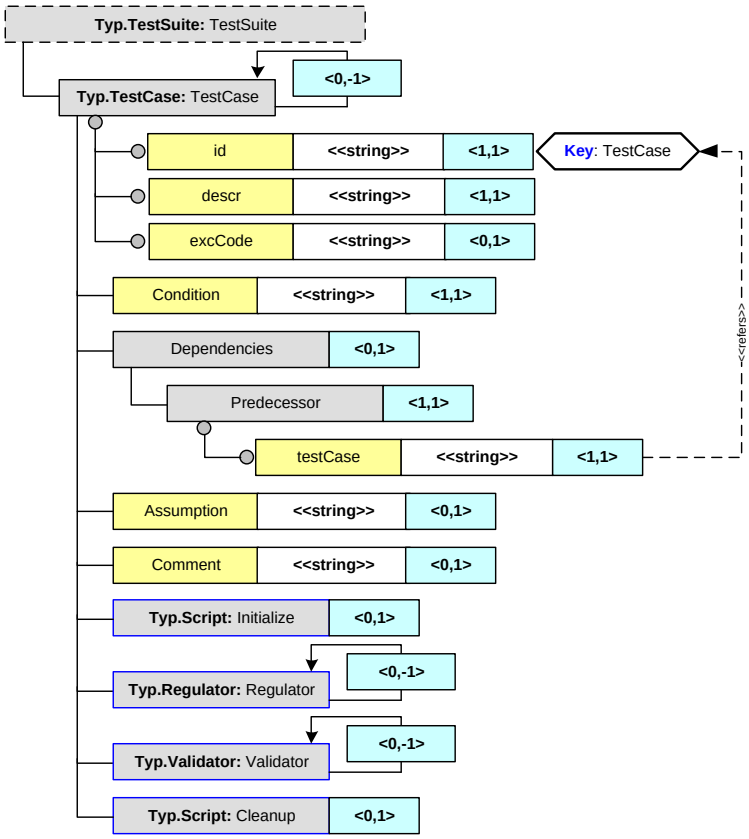


Figure 11 - Test Case parameters

Field		*Description*
TestCase *(n)*		Multiple Test Cases may be defined in the Test Suite.
	Id	Unique Key for identifying the Test Case.
	Descry	Short description for the Test Case.
	excCode	When testing exception conditions the Exception Code may be specified for reference. This distinguishes the "happy path" from the negative tests. It is good practice to check if all the exception conditions are covered in the Test Suite.
	Condition	Conditions being tested in the Test Case.
	Dependencies	Defines dependent Test Cases. If the predecessor is not run or fails, the dependent Test Case is not run.

Field	Description
Predecessor *(n)*	Multiple predecessors may be defined upon which the Test Case is dependent.
testCase	Predecessor Test Case identifier.
Assumption	Assumptions for the Test Case.
Comment	Comments for the Test Case.
Initialize	Initialization scripts for the Test Case are defined in this section.
Regulator *(n)*	Regulator definition for sending Test Data.
Validator *(n)*	Validator definition for validating Results of a test.
Cleanup	Cleanup scripts for the Test Case are defined in this section.

5.4.1. Regulator

Regulators send Test Data to one or multiple Destinations. Regulators run under different Regulation Modes. There is no limitation on multiple Regulators run in different modes in the same or different Test Cases. Each Regulator may be individually scheduled based on specific time or with delays. For more information, please refer to the 4.4.3 section.

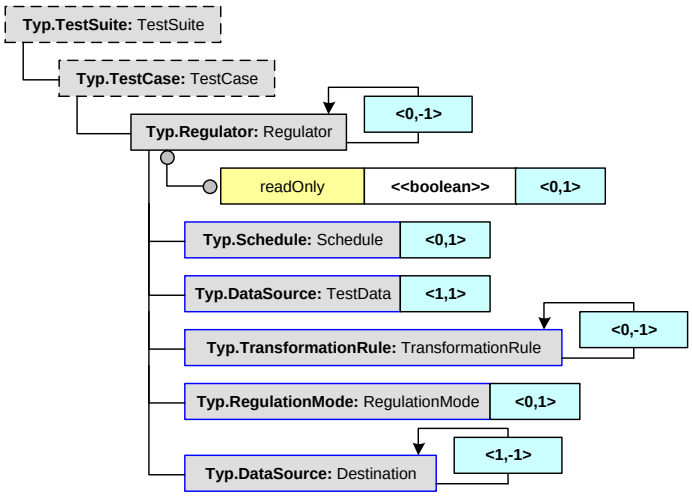


Figure 12 – Regulator parameters

Field	*Description*
Regulator *(n)*	Multiple Regulators may be defined for sending Test Data under different modes.
readOnly	By default the Regulator uses the Test Data in Read Only mode. The Test Data may be destructively read by setting readOnly to false.
Schedule	Schedule for the Regulator.
TestData	Test Data may be any supported Data Source.
TransformationRule *(n)*	Transformation Rules for transforming Test Data before sending to the Destination. Transformations may be applied at the Regulator scope or at the Destination scope. When applied at this level, the transformations are applied to the data going to each Destination that sets **inheritTR = true**. By default inheritance of transformations are turned off.
RegulationMode	Different Regulation modes may be configured for the Regulator. When the mode is unspecified, it implies that the **mode = unregulated**, i.e. the Test Data is sent to the Destination as soon as it is received after applying the transformations. For more controlled regulation, modes may be specified.
Destination *(n)*	Multiple Destinations may be specified for the Regulator. The same Test Data is sent to all the Destinations. However, different transformations may be applied to the data before sending to the Destination.

5.4.1.1. Regulation Mode

Regulators may be configured to run under several modes. The simplest of the mode is unregulated, i.e. the messages are sent to the Destination as fast as possible from the Test Data. This mode is sufficient to perform most functional testing of data where only a few sets of data are required o test the condition. However, while performing a performance test this is not sufficient since the system would be IO-bound just to send the data to the destination affecting the performance numbers. If the Test Robot is run from a separate node at a steady load level, say 50 messages per second t a queue, a better performance number may be obtained. The different modes of regulation are explained in details below.

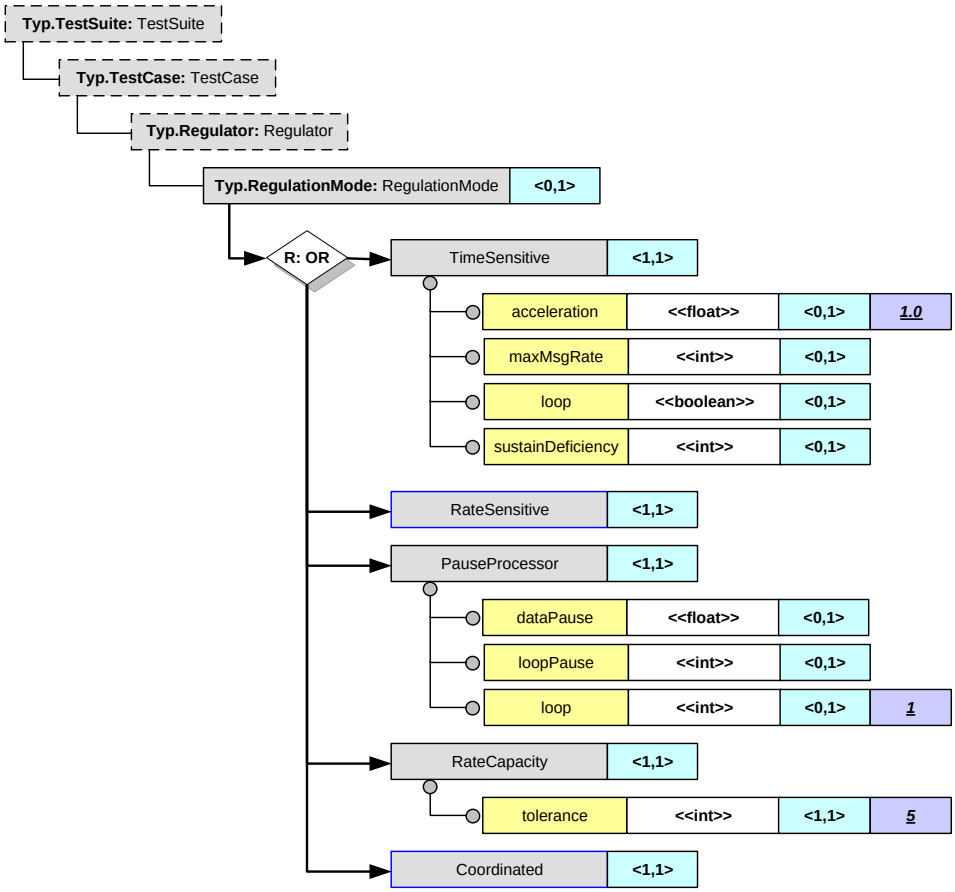


Figure 13 – Regulation Mode parameters

Field	*Description*
RegulationMode	Mode in which the Regulator is configured.
TimeSensitive	The Regulator in Time Sensitive mode runs based on how the data arrived at the system. The data must be saved with the timestamp as the file name for the data in nested directories. This is useful to simulate the exact load curve over a typical day and play it back in the new system to be migrated (not implemented in this release).
acceleration	To save time of the overall test cycle, the data may be played back at a faster rate than actual data rate, e.g. the data captured over 6 hours may be played back in 3 hours on a faster system or to test twice the capacity of the current system for future growth. In this example the acceleration will be set to 2.0.
maxMsgRate	To avoid systemic problems in an accelerated Time Sensitive test the maximum message rate at any given point may be set to a specific value and the regulation will be flat-lined when this rate is reached.
sustainDeficiency	This field indicates the number of seconds to sustain the deficiency of the message rate

Field	Description
	before quitting the test with failure. A value of -1 indicates that the deficiency may be maintained till the end of the test.
RateSensitive	The Regulator in Rate Sensitive mode sends Test Data to the Destination based on the message rate specified. The data is cached in a read-forward mechanism and the autonomic algorithm regulates the Loader and Sender thread spawning to achieve the steady rate at any given moment. The data is evenly distributed within the second as compared to spurts at the top of the second in most algorithms. The calculations are done in nano-seconds and rounded up for every second, 15 seconds, minute and 15 minutes. More on Rate Regulation later in this section.
RateCapacity	The Regulator in Rate Capacity mode may be run to determine the upper threshold of the Rate Sensitive regulation. This is used to determine the true Capacity of the system after the performance criteria is met by performing a Rate Sensitive regulation test. The performance rate criterion is usually set as the start message rate and incremented until a maximum steady state is reached. The Steady State is determined by the interval in seconds set to maintain the message rate without message accumulation or the ability to send messages within the acceptable tolerance levels (not implemented in this release).
tolerance	Tolerance is the percentage of the message rate within which the variance of the actual message rate achieved is acceptable for maintenance of a Steady State.
startMsgRate	Message Rate in messages per second at which the test starts and incremented until the full capacity of the system is reached.
rateIncrement	The increment in messages per second to determine the maximum Steady State.
steadyState	The duration in seconds to determine Steady State success or failure. If within this period of time the Steady State is not reached the test ends in failure. If Steady State is maintained for the period of time the rate is incremented to test Steady State for the next interval until the maximum Capacity is reached.
Coordinated	Coordinated Regulator sends data from the Test Data source and waits on the Coordinated Validator to validate the data before sending the next data from the source. This is used each data flowing through the system must be individually validated but a large set of data is available for test. The Regulator signals the Validator after sending the data and the Validator signals the Regulator after validating the data. More on Coordinated Regulation later in this section.
PauseProcessor	The Regulator in Pause Processing mode may be configured to wait for a specific period in seconds between each data in the Test Data source and/or between iterations of the data loop.
dataPause	Pause in seconds between successive data send from Test Data to Destination.
loopPause	Pause in seconds between successive iterations of data send from Test Data to Destination wile looping.
loop	Number of loops of the data from the Test Data source to the Destination.

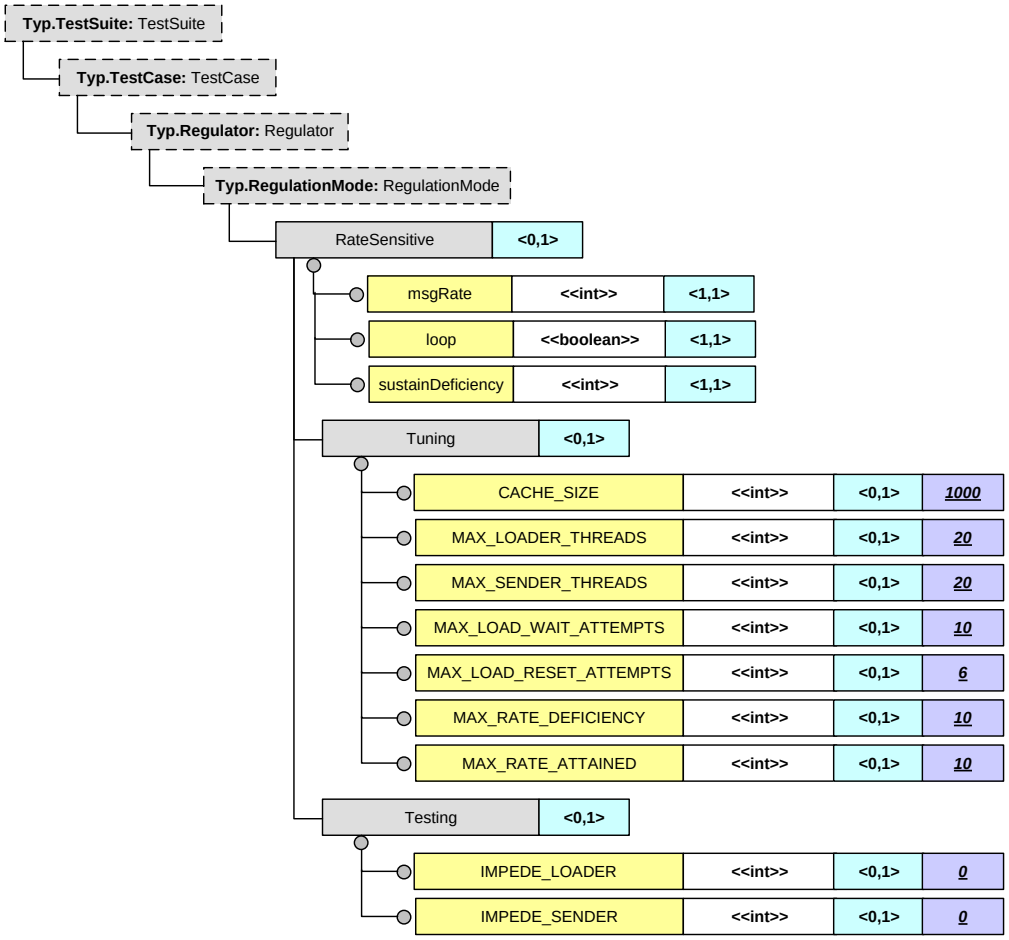


Figure 14 - Rate Sensitive parameters

Field	*Description*
RateSensitive	The Regulator in Rate Sensitive mode sends Test Data to the Destination based on the message rate specified. The data is cached in a read-forward mechanism and the autonomic algorithm regulates the Loader and Sender thread spawning to achieve the steady rate at any given moment. The data is evenly distributed within the second as compared to spurts at the top of the second in most algorithms. The calculations are done in nano-seconds and rounded up for every second, 15 seconds, minute and 15 minutes.
msgRate	Message Rate to be regulated over the test. Tolerance is added in the algorithm pessimistically so that the attained rate is slightly higher but never under the specified rate.
loop	Data looping may be enabled for the test when insufficient data is present to sustain the test over the specified duration in the Schedule and duplicate data either does not negate the test or may be transformed.
sustainDeficiency	Period of time in seconds when the test is run regardless of the deficient rate. If the test must be completed even when the rate is not reached, -1 may be specified.
Tuning	Tuning parameters for the Rate Regulation. These parameters should be used with caution. The CACHE_SIZE, MAX_LOADER_THREADS and MAX_SENDER_THREADS may be adjusted for performance and memory constraints on the system.
CACHE_SIZE	The size in number of messages (not size) stored in the read-forward cache.
MAX_LOADER_THREADS	The maximum number of threads spawned for loading the cache.
MAX_SENDER_THREADS	The maximum number of threads spawned for sending data from the cache.
MAX_LOAD_WAIT_ATTEMPTS	The maximum number of consecutive attempts to receive data from the empty cache allowed. If the number of attempts is exceeded, it is reported

Field	Description
	in the log. if the number of loader threads is less than the MAX_LOADER_THREADS then a Secondary Loader thread is spawned.
MAX_LOAD_RESET_ATTEMPTS	The maximum number of consecutive loading attempts when the cache is full. This indicates that the Secondary Loader thread spawned may be ended. In other words, it resets the Secondary Loader thread.
MAX_RATE_DEFICIENCY	The maximum number of consecutive times a rate deficiency is detected before reporting to the log. If the number of sender threads is less than the MAX_SENDER_THREADS then a Secondary Sender thread is spawned.
MAX_RATE_ATTAINED	The maximum number of consecutive times the message rate is attained or exceeded to end spawned Secondary Senders. In other words, it resets the Secondary Sender thread.
Testing	Testing parameters for the Rate Regulation should only be used to test configuration of the tool in a specific environment. This should only be used when instructed by ETR Support Team.
IMPEDE_LOADER	Time in milliseconds (10^{-3} seconds) to impede the Loader threads from loading data to the cache from the Test Data source.
IMPEDE_SENDER	Time in milliseconds (10^{-3} seconds) to impede the Sender threads from sending data from the cache to the Destination.

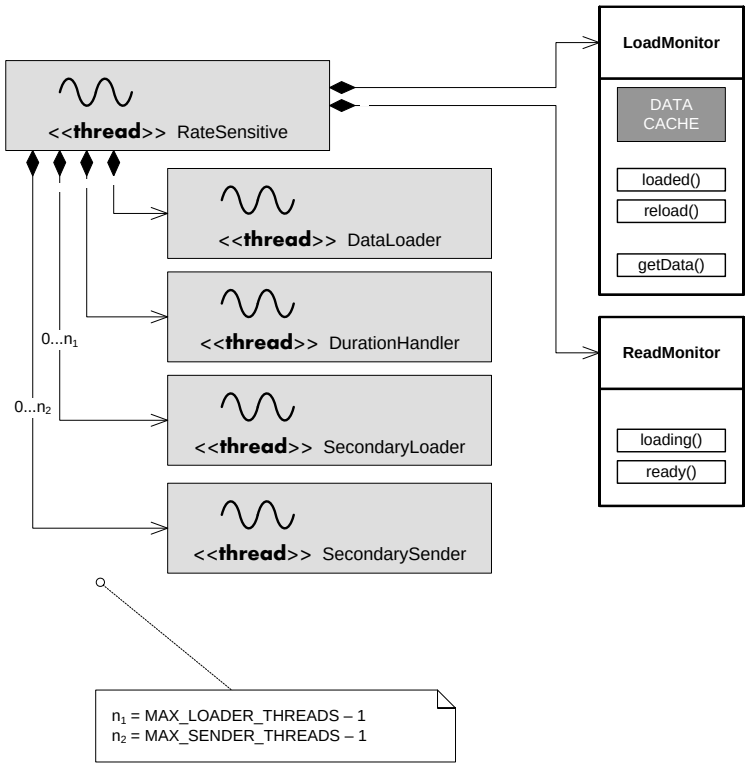


Figure 15 – Thread handling inside the Rate Sensitive Regulation

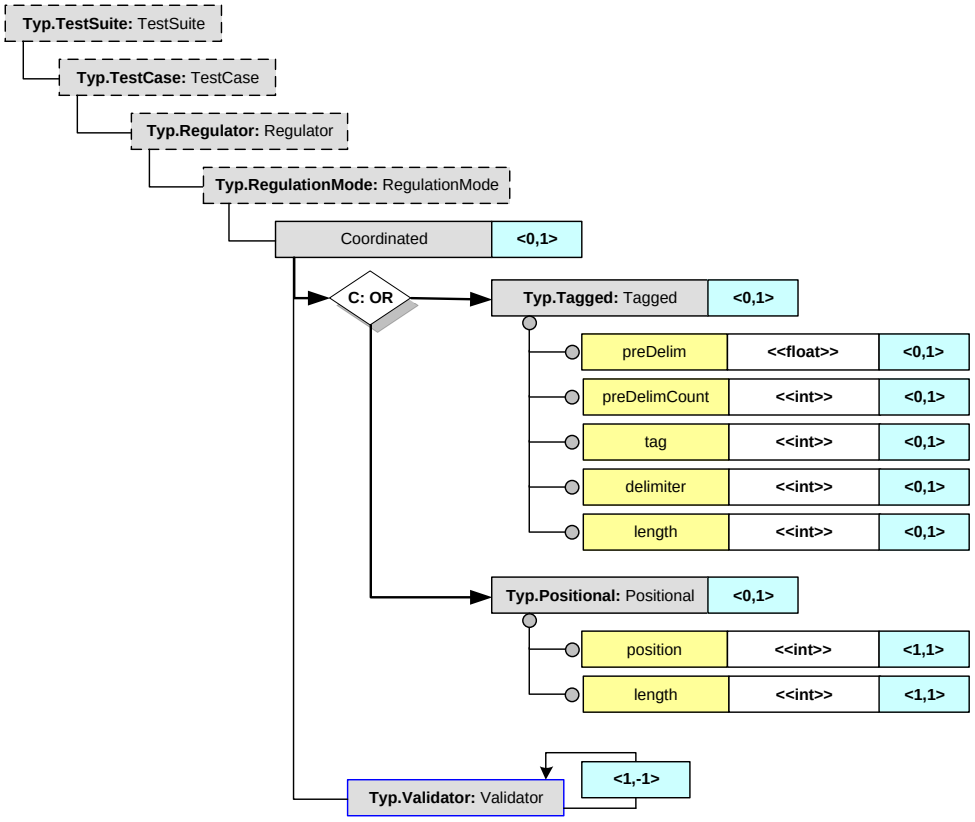


Figure 16 – Coordinated Regulation parameters

Field	*Description*
Coordinated	Coordinated Regulator sends data from the Test Data source and waits on the Coordinated Validator to validate the data before sending the next data from the source. This is used each data flowing through the system must be individually validated but a large set of data is available for test. The Regulator signals the Validator after sending the data and the Validator signals the Regulator after validating the data.
Tagged	Tagged data selected and passed to the Validator.
preDelim	Delimiter to be skipped in the selection criteria.
preDelimCount	Number of pre-delimiters to be skipped.
tag	Fixed length of the tag.
delimiter	Delimiter for the selected data. Cannot be used with the length attribute.
length	Length of the data to be selected. Cannot be used with the delimiter attribute.
Positional	Positional data selected and passed to the Validator.
position	Fixed start position of the selected data.
length	Length of the selected data.
Validator *(n)*	Multiple Validators may be coordinated with the Regulator. The parameters for the Validator are the same as an independent Validator. In the coordinated Validation mode the selected data is passed to the Validators defined and synchronized per data sent to the Destination.

5.4.2. Validator

Validators retrieve Results and compare with Expected Results to determine if the test passed or failed. Multiple Validation Modes are supported. There is no limitation on multiple Validators run in different modes in the same or different Test Cases. Each Validator may be individually scheduled based on specific time or with delays. For more information, please refer to the 4.4.3 section.

In addition to the Validation Modes, multiple Query validations may be performed from the same Validator.

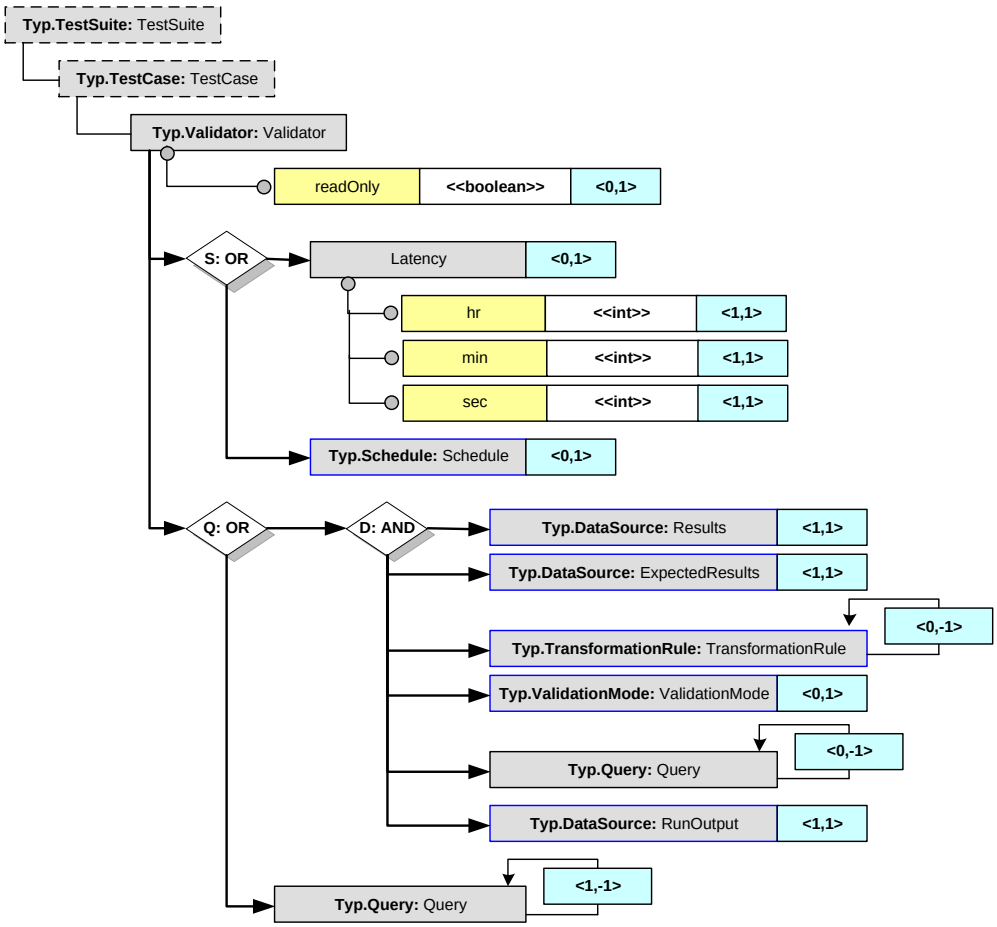


Figure 17 - Validator parameters

Field	*Description*
Validator *(n)*	Multiple Validators may be defined for validating test Results under different modes.
readOnly	By default the data is read destructively from the Results. This behavior may be altered by setting readOnly = true.
Latency	Latency is defined as the period of time to wait before the launch of the Validator from the start of the test. This feature is very useful since the application takes a finite period of time to execute the operation to be tested. Launching the Validator before the Result arrives negates the test itself.
hr	Number of hours of application latency.
min	Number of minutes of application latency.
sec	Number of seconds of application latency.
Schedule	Schedule for the Validator.
Results	Data Source where the output of the application is defined. The Results are read destructively

Field		Description
		by default and compared with the data read from the Expected Results Data Source under different validation modes. The success or failure of the Test Case depends on the validation rules.
	ExpectedResults	Data Source where the Expected Results of the test are stored before running the test. The results may be **baselined** by reversing the data stream and storing the Results into the Expected Results. This is a very convenient method of capturing the Results of the Test Run when the Results are manually determined to have passed or for setting up Test Cases for existing applications in production before making changes to the application. Thus after making the changes the same Test Case may be run (not in baseline mode) for Regression Testing after the changes.
	TransformationRule *(n)*	Transformation Rules for transforming Test Data before sending to the Destination. Transformations may be applied at the Regulator scope or at the Destination scope. When applied at this level, the transformations are applied to the data going to each Destination that sets **inheritTR = true**. By default inheritance of transformations are turned off.
	ValidationMode	Different Validation Modes may be configured for the Validator. When the mode is unspecified, it implies that the **mode = exact match**, i.e. the Results have to match exactly with the Expected Results after applying the transformations. This is the simplest form of validation and a binary compare is made for each byte.
	Query *(n)*	SQL Queries may be run alone or in addition to the Results being matched from the application output. Multiple queries may be set per Validator regardless of the Validation Mode. The comparison is made between the data in the database and the data saved in a specific format in the files. Baseline for the Query is not implemented in this release.
	RunOutput	Data Source where the Results of every Test Run is saved. The base directory for the Test Run is defined by the **appTestRunPath** attribute of the Test Suite. The path defined by the RunOutput is appended to the appTestRunPath.

5.4.2.1. Validation Mode

Validators retrieve Results and compare with Expected Results to determine if the test passed or failed. Multiple validation modes are supported and may be used in conjunction with the Transformation Rules a powerful set of tools to validate the Test Results.

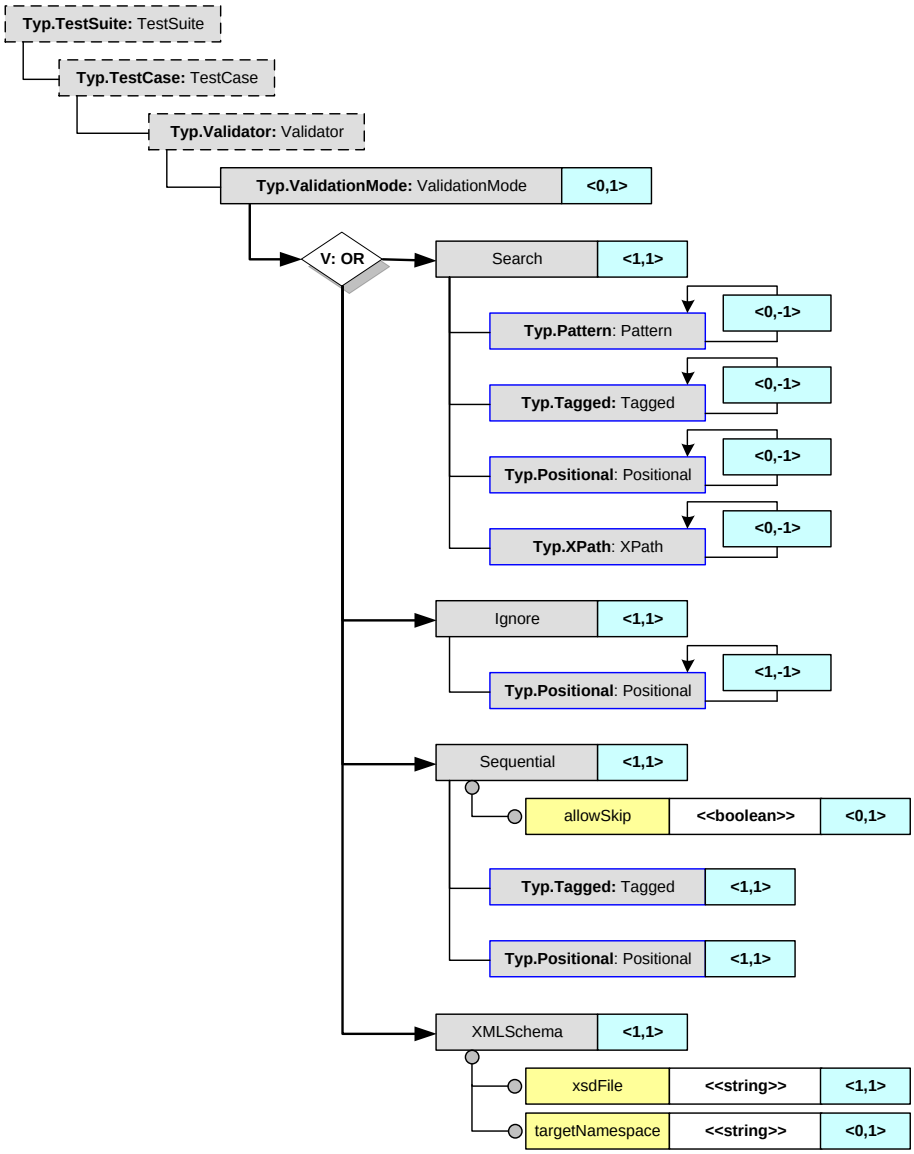


Figure 18 – Validation Mode parameters

Field	*Description*
ValidatorMode	Different Validation Modes may be configured for the Validator. When the mode is unspecified, it implies that the **mode = exact match**, i.e. the Results have to match exactly with the Expected Results after applying the transformations. This is the simplest form of validation and a binary compare is made for each byte.
Search	The Search Validation Mode is used to search for specific parts of the data. Multiple types of searches may be defined as its children. The validation is successful if all the searches are

Field	Description
	successful.
Pattern *(n)*	Regular Expression Pattern matching.
Tagged *(n)*	Tag-Delimited data matching.
Positional *(n)*	Fixed position and length of the portion of data is matched.
XPath *(n)*	If the data is XML, then XPath to the data location may be defined and matched.
Ignore	The Ignore Validation Mode is used to search for specific parts of the data and ignored. The rest of the data must match exactly. Multiple ignore portions may be defined as its children. The validation is successful if after eliminating all theses portions of data, the rest of te data is exactly matched.
Positional *(n)*	Fixed position and length of the portion of data is matched.
Sequential	Data Sequence is matched. This is used to check if a counter in the data is following the right sequence. This is needed in testing of thread synchronization logic for multi-threaded applications.
allowSkip	By default the Sequence is strictly matched by incrementing the number by 1 and starting from the first counter value received in the test.
Tagged *(n)*	Tag-Delimited data matching.
Positional *(n)*	Fixed position and length of the portion of data is matched.
XMLSchema	XML Schema to validate the XML output format.
xsdFile	XML Schema file location.
targetNamespace	If specified the schema's target namespace is also validated.

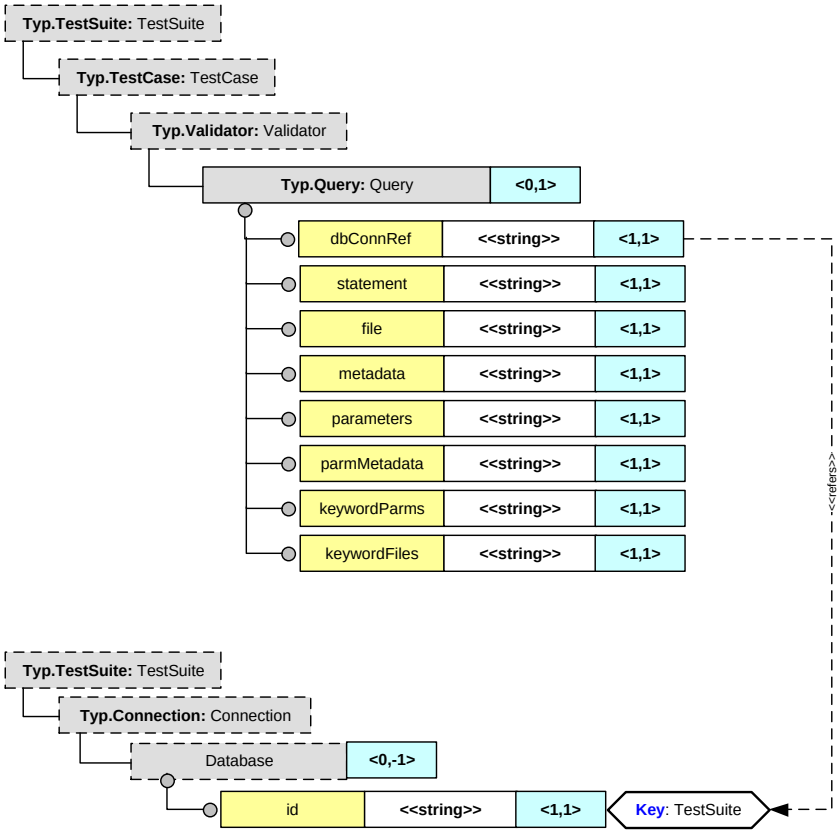


Figure 19 – Query validation parameters

Field	*Description*
Query *(n)*	SQL Queries may be run alone or in addition to the Results being matched from the application output. Multiple queries may be set per Validator regardless of the Validation Mode. The comparison is made between the data in the database and the data saved in a specific format in the files. Baseline for the Query is not implemented in this release.
dbConnRef	Database Connection Reference refers to the id for the database connection parameters. The parameters defined for connection to the database is globally defined in the Connection section.
statement	SQL query statement for data selection.
file	Expected results file to be validated against. The data is saved in a comma-separated (,) format. If data contains comma (,) or if the data is binary, the data is saved in a file in the subdirectory and the reletive file URL (file://) is saved in the data file. The relative filename generated is in the following format: **<column name>/<column name>_<%010d - row number>.dat**
metadata	Metadata for the table columns in the query. Metadata is discovered if available for the database and available. If metadata is not available, a comma-separated (,) may be provided.
parameters	Parameters to be passed to the prepared statement.
parmMetadata	Metadata for the parameters in the statement. Parameter Metadata is discovered if available for the database and driver. If metadata is not available, a comma-separated (,) may be provided.
keywordParms	Keywords supported by the Test Robot may be supplied and replaced in the parameters. Keywords are explained in details in the Transformation section.
keywordFiles	Keywords may receive data from a comma-separated (,) file and replace the keyword from the file before applying the transformation on the parameter. Please refer to the Transformation section for details.

The Metadata is based on SQL to Java type mapping recommendations from Oracle
http://docs.oracle.com/javase/1.5.0/docs/guide/jdbc/getstart/mapping.html

Support data typed defined in **java.sql.Types**:
BIGINT, BINARY, BIT, BOOLEAN, CHAR, DATALINK, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, LONGVARBINARY, LONGVARCHAR, NUMERIC, DOUBLE, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, VARBINARY, VARCHAR.

Unsupported data typed defined in **java.sql.Types**:
ARRAY, BLOB, CLOB, DISTINCT, JAVA_OBJECT, REF, STRUCT.

5.4.3. Scheduler

Both Regulators and Validators can be individually scheduled. The internal Scheduler assigns Regulator or Validator tasks based on the Start and End parameters provided. If no Start parameters are provided, the Scheduler starts the task as soon as the Test Case is initialized. The Scheduler always waits for the Initiator to complete. If no End parameters are specified, the Scheduler lets the task run till the end.

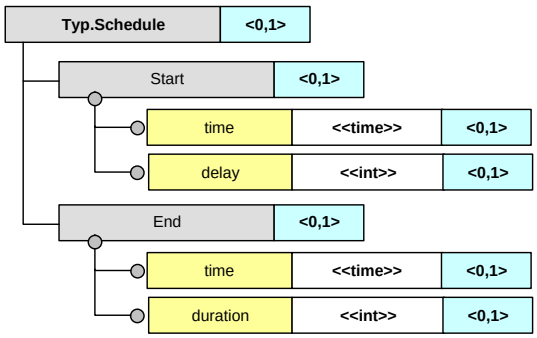


Figure 20 – Scheduler parameters

Field	*Description*
Schedule	Schedule parameters for Regulator or Validator tasks.
Start	Schedule Start configuration.
time	Specific time of the day to start the schedule.
delay	Delay in seconds after the Initiator for the Test Case has completed to start task.
End	Schedule End configuration.
time	Specific time of the day to end the schedule.
duration	Duration in seconds after the start of the task.

5.4.4. Transformation Rule

Transformation Rules may be applied both in Regulators and Validators. Regulators may require transforming Test Data before sending to the Destination. This may be for various reasons. The Test Data may have date fields that may require setting to the current date and time with some minutes subtracted. UUID may have to be generated for the unique keys. Certain specific values may need to be replaced for each iteration of the data being sent to the Destination. Similarly, Validators may require the data from the Results and Expected Results to be transformed before comparing them. Dates may be replaces with *** to make comparison easier than having to come up with some very complex Ignore conditions. Ignore conditions in the tool only support Positional for this reason.

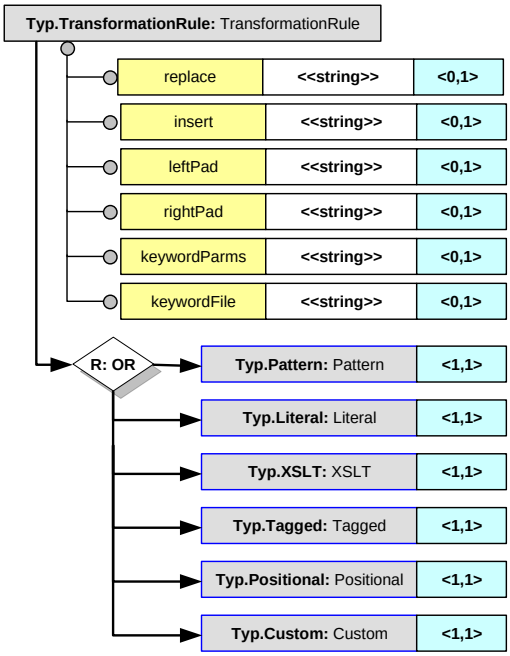


Figure 21 – Transformation Rule parameters

Field		*Description*
Transformation Rule		Rule for transforming data.
	replace	Replacement string. The string may contain Built-in Keywords (**$<keyword>.**) or Keyword Parameters (**$[<keyword>]**) defined. The selected part of the data is replaced with the string after the resolution of all the Keywords. There is no limitation in the number of times the same or different Keyword may be used in the string. It may be used along with the insert attribute.
	insert	Insert string. The string may contain Built-in Keywords (**$<keyword>.**) or Keyword Parameters (**$[<keyword>]**) defined. The string after the resolution of all the Keywords is inserted at the beginning of the selection. There is no limitation in the number of times the same or different Keyword may be used in the string. It may be used along with the replace attribute.
	leftPad	Left Padding string. This is only applicable for Positional rule. The string pattern is repeated till the field length is reached.
	rightPad	Right Padding string. This is only applicable for Positional rule. The string pattern is repeated till the field length is reached.
	keywordParms	Keyword Parameters may be defined as a comma-separated string. One or more of the Keywords defined may be used in the **Insert** or **Replace** strings. The Keyword File record is incremented per data sent and replaced in the insert/replace string and then replaced in the data based on the Rule.
	keywordFile	Keyword File works in conjunction with Keyword Parameter. Keyword Parameters essentially define the metadata for the Keyword File. The file is a comma-separated value of the

Field		Description
		parameter to be inserted/replaced in the data. The file record pointer is incremented per data sent. The record terminator is '\n' in UNIX/LINUX and '\n\r' in Windows systems.
	Pattern	Regular Expression Pattern search.
	Literal	Literal String search.
	XSLT	XML Style-sheet Language Transformation applied to the data before sending to the target Data Source. This is only applicable for XML data and other attributes may not be used with this Transformation Rule (not implemented in this release).
	Tagged	Tag-Delimited data search.
	Positional	Fixed position and length of the portion of data is searched.
	Custom	Custom Transformation Rule plug-ins may be added to the tool by implementing the ***IDataTransformation*** interface (not implemented in this release).

5.4.4.1. Keyword

The following table describes the different keywords that may be defined in the insert or replacement strings. There is no limitation of the number of times the same or different keywords may be repeated in the string. The values are synchronized within the same string so that it can be referred multiple times, e.g. Counter, Current Timestamp, etc.

Keyword	*Description*
$Yr.	2-digit current year.
$Year.	4-digit current year.
$Mon.	2-digit current month.
$Day.	2-digit current day of the month.
$hr.	2-digit current hour.
$min.	2-digit current min.
$sec.	2-digit current second.
$DateFormat('<pattern>').	Date and time formatting based on the pattern provided within single-quotes ('). For more information about date time patterns please refer to: http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html#SimpleDateFormat(java.lang.String)
$DateChange('<calculation>', '<pattern>').	The date Change function is an extension of the Date Format functionality to perform simple data time arithmetic with the current date and time and return the value in a formatted pattern. The calculation string is of the following characteristics: <calculation> = (+ | -)<number>(y | M | d | h | m | s | S) where y = year M = month d = day h = hour m = minute s = second S = millisecond The addition (+) or subtraction (-) operator must immediately be followed by the integer and one of the operator actions (year, month, etc.). Any sequence of characters not following the calculation pattern is ignored. There is no limitation on the number of times the calculation operation may be repeated. The date time formatting pattern follows the same rules as the ***DateFormat*** keyword. Please refer to the date time formatting documentation at: http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html#SimpleDateFormat(java.lang.String)
$Counter.	The counter is a sequential number and incremented by 1 for every use in a string. The counter is not incremented within the same string so that it may be referred to multiple times.
$UUID.	The UUID is generated every time the function is called even within the same string.
$[<parm>]	The parameter is defined as comma separated values with the **keywordParms** attribute and parameters are read at runtime from the file defined by the **keywordFile** attribute in the same order as the defined list. All the parameters defined in the keywordParms must be defined in the keyword file as comma separated values. Each line in the file is considered to be a record. Every time the Insert or Replace is called, the record is incremented and the parameter is read from the record. The parameter can be defined any number of times or never in the insert or replace string. So the same file and the parameter list may be used multiple times while a subset of the parameters is used. The insert/replace string is replaced with the parameter from the file and then applied to the data based on the Transformation Rule. e.g. **replace**="$[LATITUDE]" **keywordParms**="LATITUDE,LONGITUDE,SPEED" **keywordFile**="data/data.01/parm" file data: 100.04,120.44,82 109.99, 200.54,91

5.4.4.2. Transformation Rule: Pattern

Regular Expressions may be used to search strings within the data and replaced with the string specified in the replace attribute. The insert attribute is not compatible with this transformation.

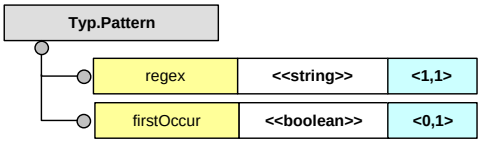


Figure 22 – Regular Expression Pattern Transformation

Field		*Description*
Pattern		Regular Expression Pattern search.
	regex	Regular Expression string based on the Java supported expression syntax.
	firstOccur	Optional parameter to indicate that only the first occurrence is to be replaced when set to true. If not specified or set to false, all occurrences are replaced.

5.4.4.3. Transformation Rule: Literal

This transformation rule is used to search for literal string values within the data. Both the insert and the replace attributes are valid for this transformation. The first occurrence of the sting is only checked. The use of this transformation is mainly to mark the data with unique markers and replace it at runtime while the data is sent to the Destination.

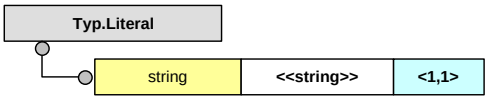


Figure 23 – Literal String Transformation Rule

Field	*Description*
Literal	Literal String search.
string	String to search in the data.

5.4.4.4. Transformation Rule: Tagged

Most Tagged-Delimited structures can be parsed by the options provided by the Tagged Transformation Rule for the purpose of transforming the data for testing. The options provide a simple but powerful method of skipping over a specified number of delimiters, then searching for a tag followed by an offset length. The item delimiter can then be specified or a length to determine the length of the searched item in the data.

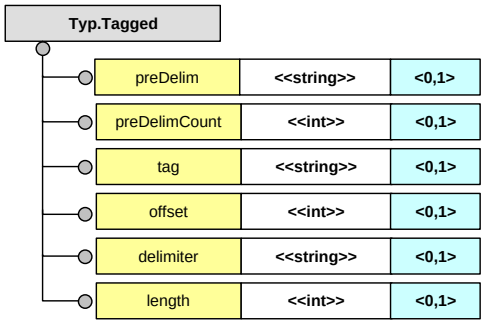


Figure 24 – Tagged-Delimited Transformation Rule

Field	*Description*
Tagged	Regular Expression Pattern search.
preDelim	Prefixed Delimiters to be skipped.
preDelimCount	Number of Prefixed Delimiters to be skipped. This attribute is mandatory if **preDelim** is specified.
tag	The Tag string to identify the item tag.
offset	The Offset number of bytes from the end of the tagged string. This attribute is only valid when **tag** is specified.
delimiter	Item Delimiter for determining the end of selection. This attribute is mutually exclusive with the **length** attribute.
length	Item Length for determining the end of selection. This attribute is mutually exclusive with the **delimiter** attribute.

5.4.4.5. Transformation Rule: Positional

Data elements based on positions may be specified using the position and length from the beginning of the data. The first position is 0.

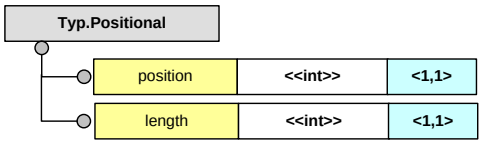


Figure 25 – Positional Transformation Rule

Field	*Description*
Positional	Fixed position and length of the portion of data is searched.
position	Position pointer for the data element.
length	Length of the data element.

5.4.5. Data Source

Data Sources are an abstracted entity that determines the source and destination for any test configuration. The abstraction of the Data Source layer makes the tool adaptable to a wide range of protocols that it may be extended to while preserving the functionality of the core product. In this version of the tool the File Source and Queue Source is implemented by extending the Data Source abstract class.

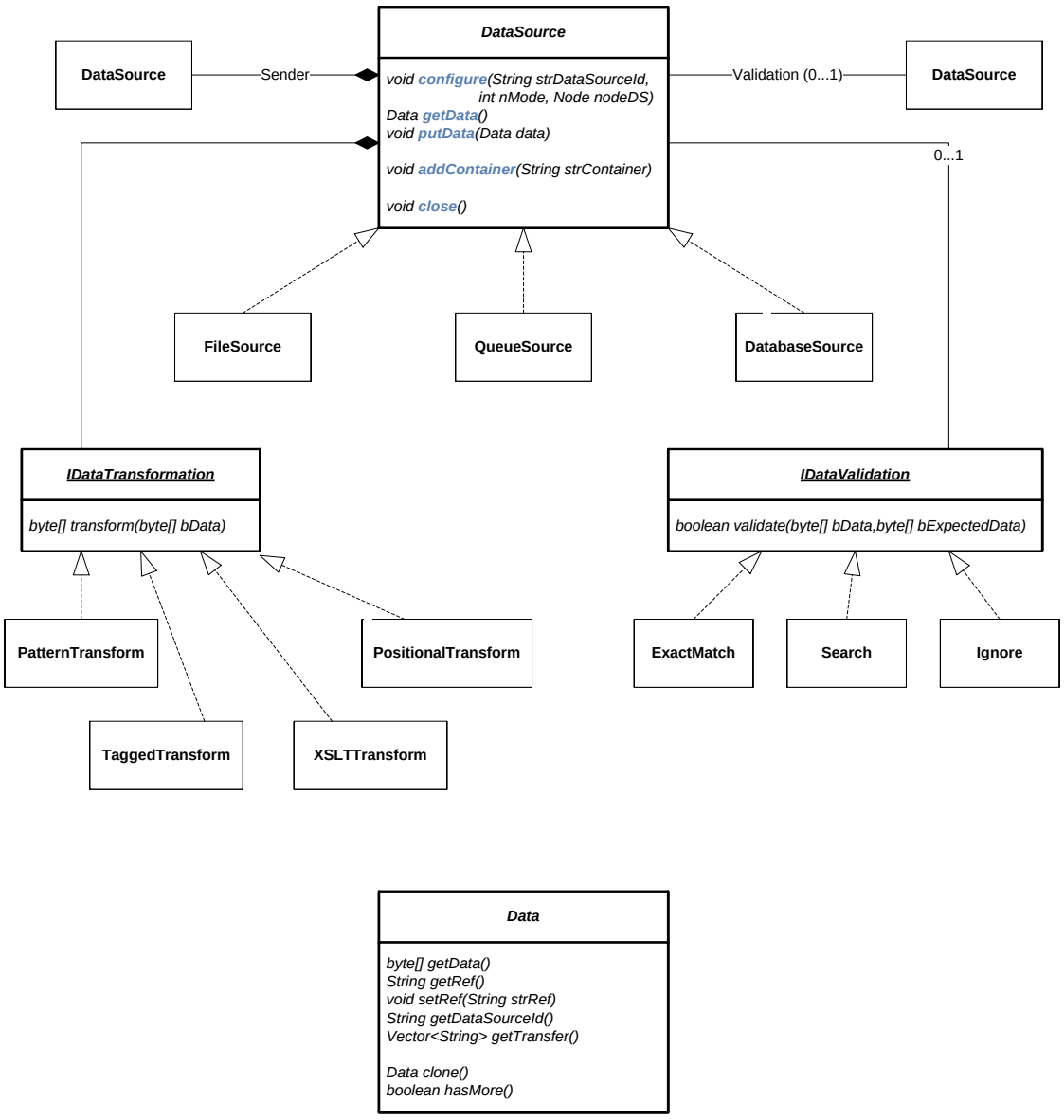


Figure 26 – Class Diagram of Data Source and related classes

The above diagram illustrates the DataSource abstract class along with some of the related classes. The abstract functions for the Data Source are shown in the diagram. The extensibility of the framework is restricted to IBM. However, an extensibility engagement can be conducted to extend the framework for custom requirements.

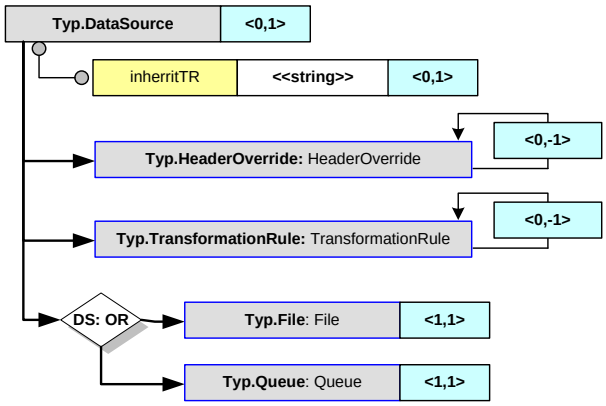


Figure 27 – DataSource

The abstracted Data Source may be configured for all the XML elements that use this data type, viz. **TestData**, **Destination**, **Results**, **ExpectedResults** and **RunOutput**.

Field	*Description*
DataSource	Data Source reference for TestData, Destination, Results, ExpectedResults and RunOutput.
inheritTR	Inherit the transformation rules from the Regulator or the Validator that contains the Data Source.
HeaderOverride	Parameters to override the data header either created with default values or from a supported format, e.g. File format of MQMD with header MQMDHeader.
TransformationRule	Transformation rules that apply to the Regulator and Validator also apply to every Data Source. This parameter may be used for different transformation requirements for each of the Destinations when using multiple Destinations from a single Test Data.
File	File Data Source gets and puts data to and from a File.
Queue	Queue Data Source gets and puts data to and from a Queue.

The Header Override supported in this version is only for **MQMD**. However, the framework may be extended to other header types in the future. A detailed list of the **MQMD Header** is documented below. The table shows the mapping to the **MQMD** names in the product documentation and a short description of the fields.

The Transformation Rules are same that may be applied at the Regulators and the Validators. The **inheritTR** flag is used to inherit the Transformation Rules from the Regulators and Validators and combined with those defined for the Data Source. For more details of Transformation Rules, please refer to **4.4.4 Transformation Rule**.

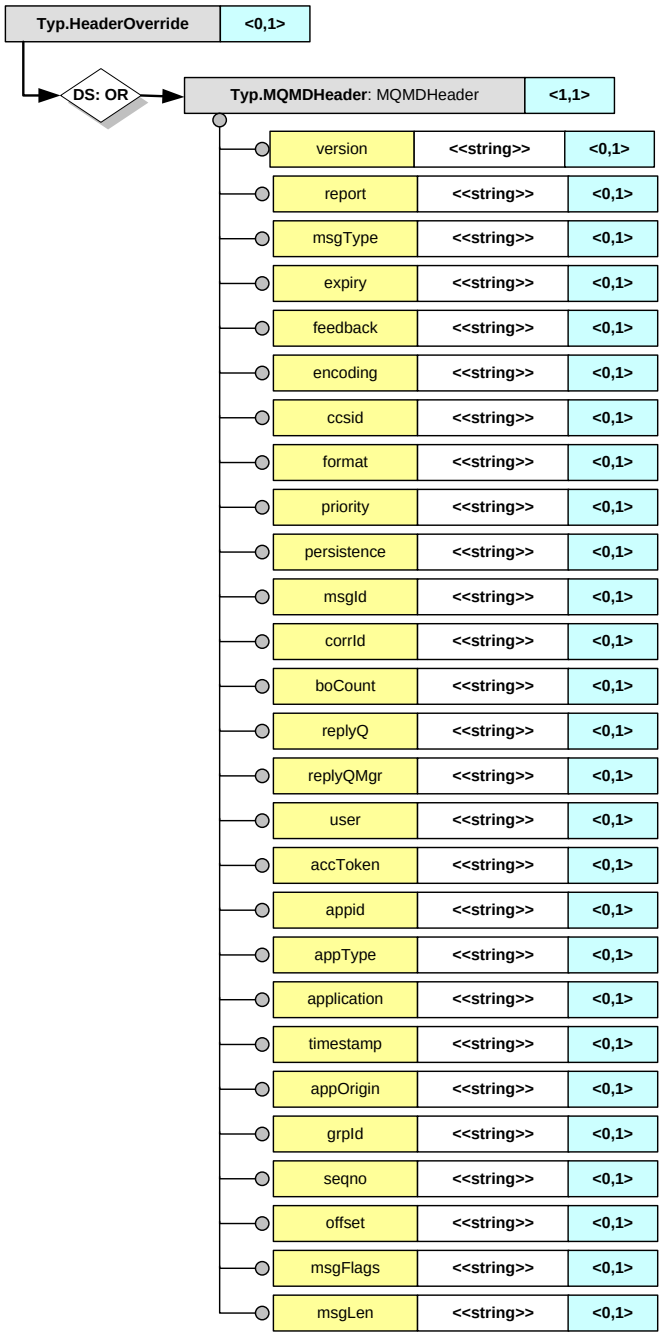


Figure 28 - Header Override

The information about each field is documented in the table below.

Field	Description
HeaderOverride	
MQMDHeader	
version	MQMD.Version: must be 2 for WMQ 6.0+
report	MQMD.Report: A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The Report field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested: • Exception • Expiration • Confirm on arrival (COA) • Confirm on delivery (COD) • Positive action notification (PAN) • Negative action notification (NAN)
msgType	MQMD.MsgType: This indicates the type of the message. • Datagram • Request • Reply • Report
expiry	MQMD.Expiry: message expiry in 10th of a second
feedback	MQMD.Feedback: The Feedback field is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message.
encoding	MQMD.Encoding: This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.
ccsid	MQMD.CodedCharSetId: This field specifies the character set identifier of character data within the message body. If the CharacterSet property is set to MQCCSI_Q_MGR, the code page for the current locale is used for character conversion in the WriteString method. For server applications, the code page used is the code page of the queue manager; for client applications, the code page is the default current locale code page. For client applications, MQCCSI_Q_MGR is filled in, based on the locale of the client rather than the one on the queue manager. The exception to that rule is when you put a message to an IMS Bridge queue; what is returned, in the CodedCharSetId field of MQMD, is the CCSID of the queue manager.
format	MQMD.Format: A name which indicates the nature of the data in the message. It is set by the sender. You can use your own format names, but names beginning with the letters "MQ" have meanings that are defined by the queue manager. The queue manager built-in formats are: • MQC.MQFMT_NONE • MQC.MQFMT_ADMIN • MQC.MQFMT_COMMAND_1 • MQC.MQFMT_COMMAND_2 • MQC.MQFMT_DEAD_LETTER_HEADER • MQC.MQFMT_EVENT • MQC.MQFMT_MD_EXTENSION • MQC.MQFMT_PCF • MQC.MQFMT_STRING • MQC.MQFMT_TRIGGER • MQC.MQFMT_XMIT_Q_HEADER
priority	MQMD.Priority: The priority of the message and may be set to an integer between 0 and 9. The default value is 0.

Field			Description
		persistence	MQMD.Persistence: This indicates whether the message survives system failures and restarts of the queue manager.
		msgId	MQMD.MsgId: This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is not converted between character sets when the message flows from one queue manager to another. The id may be assigned in several methods and the value is automatically padded. The following prefixes are used. Hexadecimal: 0x Decimal number: 0n String assignment: (no prefix)
		corrId	MQMD.CorrelId: The CorrelId field is property in the message header that may be used to identify a specific message or group of messages. This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is not converted between character sets when the message flows from one queue manager to another. The id may be assigned in several methods and the value is automatically padded. The following prefixes are used. Hexadecimal: 0x Decimal number: 0n String assignment: (no prefix)
		boCount	MQMD.BackoutCount: This is a count of the number of times that the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It helps the application to detect processing errors that are based on message content. Every time a transaction failed and backed out to the queue the backout count of the MQMD is incremented. When the backout count matches the backout threshold set on the queue, then the message is eligible to be send to the backout queue specified on the queue. However, this is not an action that the MQ takes but is left to the application to handle. The Message Broker takes advantage of the parameters and takes action to send the message to the backout queue.
		replyQ	MQMD.ReplyToQ: This is the name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages. The name is the local name of a queue that is defined on the queue manager identified by ReplyToQMgr. This queue must not be a model queue, although the sending queue manager does not verify this when the message is put.
		replyQMgr	MQMD.ReplyToQMgr: This is the name of the queue manager to which to send the reply message or report message. ReplyToQ is the local name of a queue that is defined on this queue manager. If the ReplyToQMgr field is blank, the local queue manager looks up the ReplyToQ name in its queue definitions. If a local definition of a remote queue exists with this name, the ReplyToQMgr value in the transmitted message is replaced by the value of the RemoteQMgrName attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the ReplyToQMgr that is transmitted with the message is the name of the local queue manager.
		user	MQMD.UserIdentifier: This is part of the identity context of the message. UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

Field			Description
		accToken	MQMD.AccountingToken: This is the accounting token, part of the identity context of the message. AccountingToken allows an application to charge appropriately for work done as a result of the message. The queue manager treats this information as a string of bits and does not check its content. The id may be assigned in several methods and the value is automatically padded. The following prefixes are used. Hexadecimal: 0x Decimal number: 0n String assignment: (no prefix)
		appid	MQMD.ApplIdentityData: This is part of the identity context of the message. ApplIdentityData is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.
		appType	MQMD.PutApplType: This is the type of application that put the message, and is part of the origin context of the message.
		application	MQMD.PutApplName: This is the name of application that put the message, and is part of the origin context of the message.
		timestamp	MQMD.PutDate and MQM.PutTime: This is the date and time when the message was put, and is part of the origin context of the message. The date time format for this field is: **yyyy-MM-dd HH:mm:ss.SSS z**
		appOrigin	MQMD.ApplOriginData: This is part of the origin context of the message. ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted.
		grpId	MQMD.GroupId: This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. GroupId is also used if segmentation is allowed for the message. In all these cases, GroupId has a non-null value, and one or more of the following flags is set in the MsgFlags field: • MQMF_MSG_IN_GROUP • MQMF_LAST_MSG_IN_GROUP • MQMF_SEGMENT • MQMF_LAST_SEGMENT • MQMF_SEGMENTATION_ALLOWED On the MQPUT call, MQPMO_LOGICAL_ORDER is specified. On the MQGET call, MQMO_MATCH_GROUP_ID is not specified.
		seqno	MQMD.MsgSeqNumber: This is the sequence number of a logical message within a group. Sequence numbers start at 1, and increase by 1 for each new logical message in the group, up to a maximum of 999 999 999. A physical message that is not in a group has a sequence number of 1.
		offset	MQMD.Offset: This is the offset in bytes of the data in the physical message from the start of the logical message of which the data forms part. This data is called a segment. The offset is in the range 0 through 999 999 999. A physical message that is not a segment of a logical message has an offset of zero.
		msgFlag	MQMD.MsgFlags: MsgFlags are flags that specify attributes of the message, or control its processing. MsgFlags are divided into the following categories – Segmentation flags and Status flags. Segmentation: • MQMF_SEGMENTATION_INHIBITED • MQMF_SEGMENTATION_ALLOWED Status: • MQMF_MSG_IN_GROUP

Field			Description
			• MQMF_LAST_MSG_IN_GROUP • MQMF_SEGMENT • MQMF_LAST_SEGMENT
		msgLen	MQMD.OriginalLength: This field is relevant only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, or the length of the data in the report message.

5.4.5.1. File Source

The File Source is an implementation of a Data Source to provide File handling support. The path specified for the file may relative or absolute depending upon the attribute. The path may point to either a file or a directory and the tool detects it and acts upon a single file or a group of files from the directory. By default the Test Path or the Test Run Path is prefixed to the path specified unless the absolute attribute is set to true.

If the directory specified contains sub-directories it is indexed with relative path into an **.index** file. Once the index is created it is used for subsequent references to the file. For re-indexing, when the contents have changed, remove the **.index** file or run the tool with the **–index** option. For re-running the test when it failed in the middle of a long run, the **–rerun** option may be used. This is really important when testing an application that crashes due to data conditions and have to be run with a large dataset (millions of files) to harden the code.

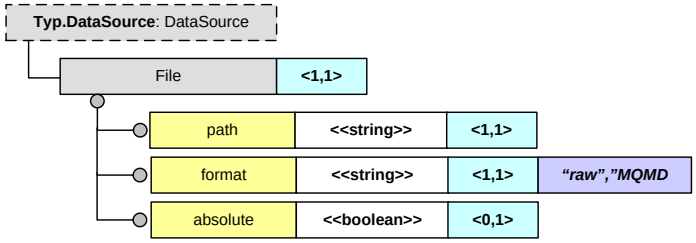


Figure 29 - File Source

Field		*Description*
File		File Data Source for file handling support.
	path	Path to the file or directory to read or write data to.
	format	The file may be of 2 formats, viz. raw and MQMD. Raw format implies that only data resides in the file. MQMD format implies that the file contains data prefixed with the MQMDHeader.
	absolute	Determines if the file path supplied is absolute.

5.4.5.2. Queue Source

The Queue Source is an implementation of a data Source to provide Queuing support for the tool. The Queue Manager Connection definition may be reused here.

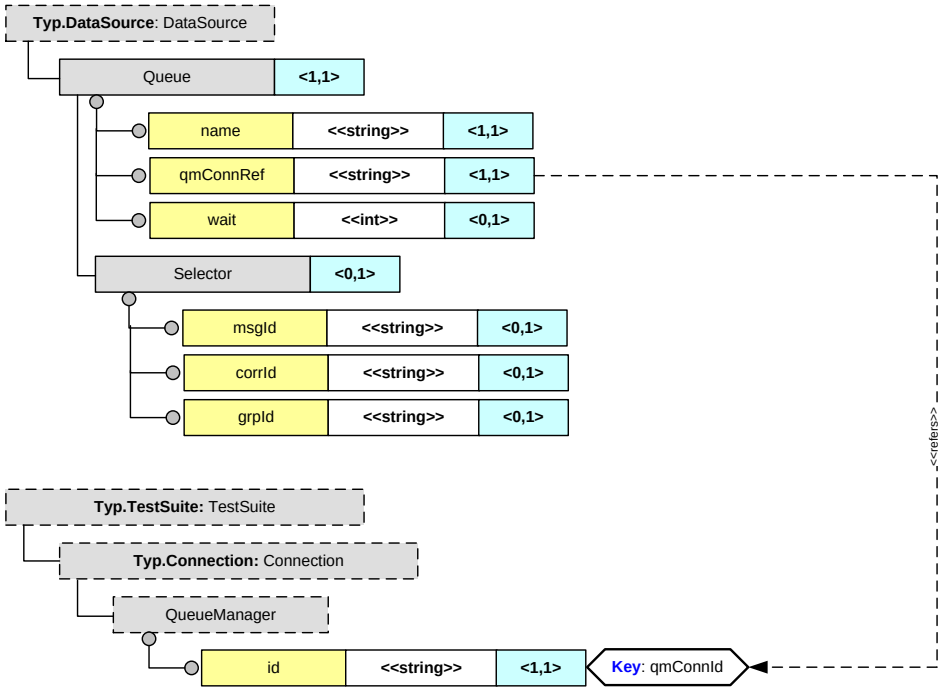


Figure 30 - Queue Source

Field	*Description*
Queue	Queue Data Source for queue handling support.
name	Name of the queue.
qmConnRef	Reference to the Queue Manager connection.
wait	Wait interval for getting a message from the empty queue.
Selector	Selection criteria for getting a message from the queue.
msgId	The message id of the message to be selected. The id may be assigned in several methods and the value is automatically padded. The following prefixes are used. Hexadecimal: 0x Decimal number: 0n String assignment: (no prefix)
corrId	The correlation id of the message to be selected. The id may be assigned in several methods and the value is automatically padded. The following prefixes are used. Hexadecimal: 0x Decimal number: 0n String assignment: (no prefix)
grpId	The group id of the message to be selected. The id may be assigned in several methods and the value is automatically padded. The following prefixes are used. Hexadecimal: 0x Decimal number: 0n String assignment: (no prefix)

6. Sample Test Cases

Sample Test Suites are provided as guidelines for different test cases that may be set up by the Enterprise Test Robot. In this release not all the options were exhaustively converted into samples and tested, but that would be our goal for subsequent releases. This will provide the ability to simply duplicate the test cases in parts or entirely and customize them for application specific requirements.

6.1. Sample 01: Supported Data Sources

The sample demonstrates the use of different Data Source combinations. The supported Data Sources in this release are File Source and Queue Source.

6.1.1. Test Data = file, Results = file, Expected Results = file

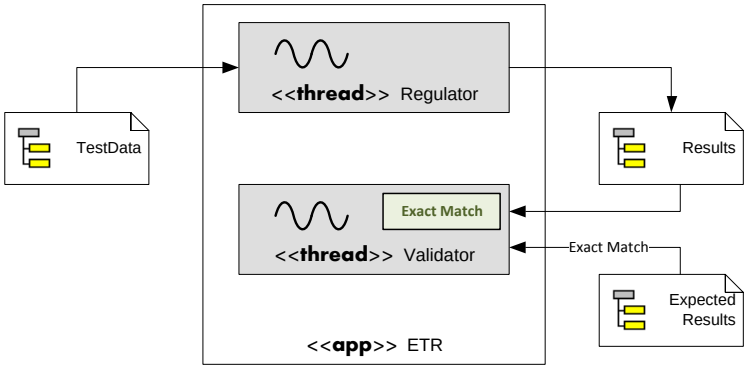


Figure 31 – All File Interactions

Test Data is saved in the file in the **data** directory and transferred to another file in the **out** as the Results by the Regulator and compared to the canned results saved in the **result** directory by the Validator.

```
<TestCase id="ETR_01_1.1_" descr="Exact Match - File to File Regulator">
        <Condition>
                - Send Test Data from a file to another file
                - Retrieve data from the output file
                - Validate data against existing data in a file
        </Condition>
        <Regulator>
                <TestData><File path="data/ETR_01/test01.txt" format="raw"/></TestData>
                <Destination><File path="out/ETR_01" format="raw"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><File path="out/ETR_01" format="raw"/></Results>
                <ExpectedResults>
<File path="results/ETR_01/test01.txt" format="raw"/>
</ExpectedResults>
                <RunOutput><File path="" format="raw"/></RunOutput>
        </Validator>
</TestCase>
```

6.1.2. Test Data = file, Results = queue, Expected Results = file

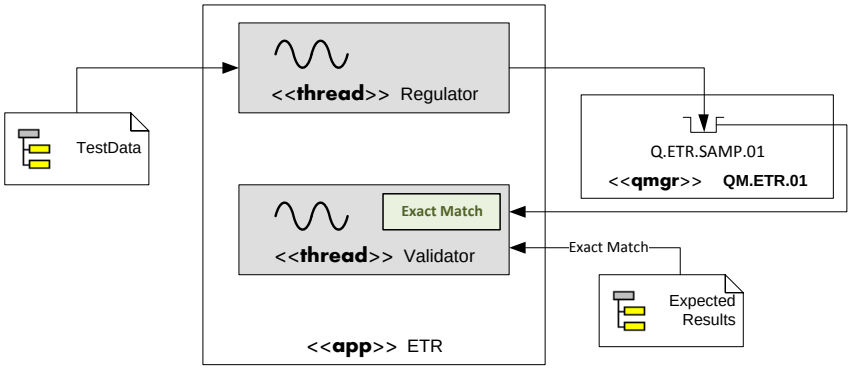


Figure 32 – Send File to Queue

Test Data is saved in the file in the **data** directory and transferred to the queue **Q.ETR.SAMP.01** as the Results by the Regulator and compared to the canned results saved in the **result** directory by the Validator.

```
<TestCase id="ETR_01_1.2_" descr="Exact Match - File to Queue Regulator">
        <Condition>
                - Send Test Data from a file to a queue
                - Retrieve data from the output queue
                - Validate data against existing data in a file
        </Condition>
        <Regulator>
                <TestData><File path="data/ETR_01/test01.txt" format="raw"/></TestData>
                <Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Results>
                <ExpectedResults>
                     <File path="results/ETR_01/test01.txt" format="raw"/>
                </ExpectedResults>
                <RunOutput><File path="" format="raw"/></RunOutput>
        </Validator>
</TestCase>
```

6.1.3. Load Queue, dependent Test Case

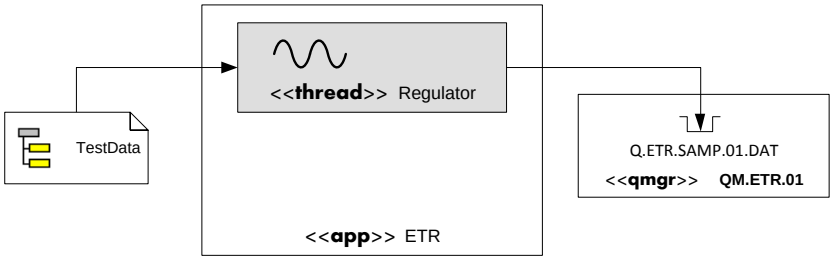


Figure 33 – Load File to Queue for subsequent tests

Test Data is saved in the file in the **data** directory and transferred to the queue **Q.ETR.SAMP.01.DAT** as the Results by the Regulator for subsequent Test Cases to use as a dependent Test Case.

```
<TestCase id="ETR_01_1.2_" descr="Exact Match - File to Queue Regulator">
        <Condition>
                - Send Test Data from a file to a queue
                - Retrieve data from the output queue
                - Validate data against existing data in a file
        </Condition>
        <Regulator>
                <TestData><File path="data/ETR_01/test01.txt" format="raw"/></TestData>
                <Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Results>
                <ExpectedResults>
                     <File path="results/ETR_01/test01.txt" format="raw"/></ExpectedResults>
                <RunOutput><File path="" format="raw"/></RunOutput>
        </Validator>
</TestCase>
```

6.1.4. Test Data = queue, Results = queue, Expected Results = file

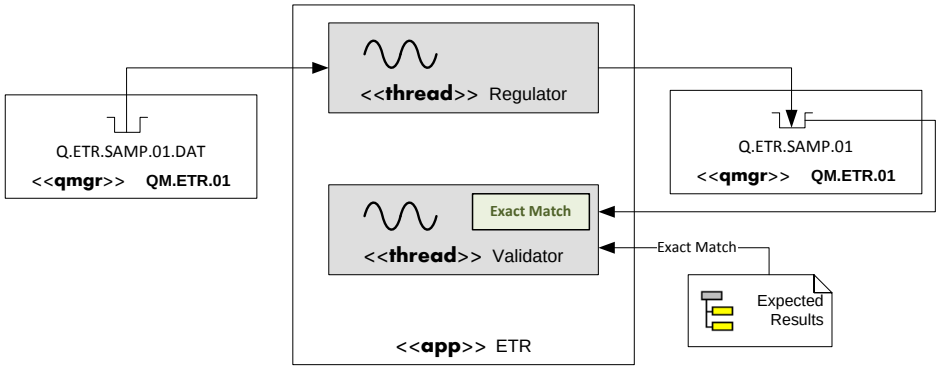


Figure 34 – Send Message to Queue

Test Data is saved in the queue **Q.ETR.SAMP.01.DAT** and transferred to the queue **Q.ETR.SAMP.01** as the Results by the Regulator and compared to the canned results saved in the **result** directory by the Validator.

```
<TestCase id="ETR_01_2.1_" descr="Exact Match - Queue to Queue Regulator">
        <Condition>
                - Send Test Data from a queue to a queue
                - Retrieve data from the output queue
                - Validate data against existing data in a file
        </Condition>
        <Dependencies>
                <Predecessor testCase="ETR_01_2.d_"/>
        </Dependencies>

        <Regulator>
                <TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01.DAT"/></TestData>
                <Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Results>
                <ExpectedResults>
                        <File path="results/ETR_01/test01.txt" format="raw"/>
                </ExpectedResults>
                <RunOutput><File path="" format="raw"/></RunOutput>
        </Validator>
</TestCase>
```

6.1.5. Test Data = queue, Results = queue, Expected Results = file

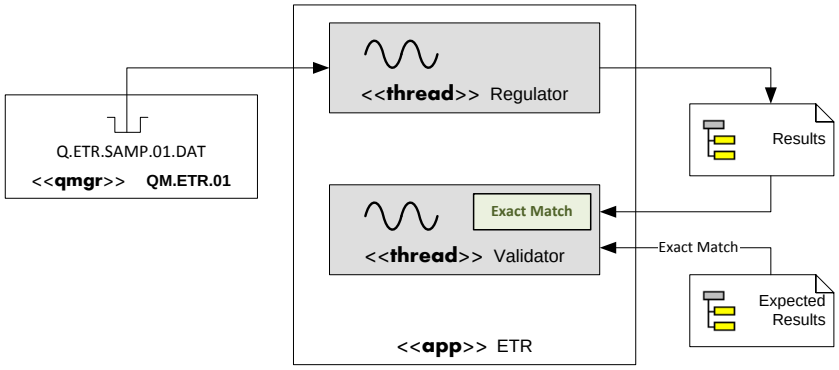


Figure 35 – Send Message to File

Test Data is saved in the queue **Q.ETR.SAMP.01.DAT** and transferred to the file **out** directory as the Results by the Regulator and compared to the canned results saved in the **result** directory by the Validator.

```
<TestCase id="ETR_01_2.2_" descr="Exact Match - Queue to File Regulator">
        <Condition>
                - Send Test Data from a queue to a file
                - Retrieve data from the output queue
                - Validate data against existing data in a file
        </Condition>
        <Dependencies>
                <Predecessor testCase="ETR_01_2.d_"/>
        </Dependencies>

        <Regulator>
                <TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01.DAT"/></TestData>
                <Destination><File path="out/ETR_01" format="raw"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><File path="out/ETR_01" format="raw"/></Results>
                <ExpectedResults>
                     <File path="results/ETR_01/test01.txt" format="raw"/>
                </ExpectedResults>
                <RunOutput><File path="" format="raw"/></RunOutput>
        </Validator>
</TestCase>
```

6.1.6. Test Data = queue, Results = queue, Expected Results = queue

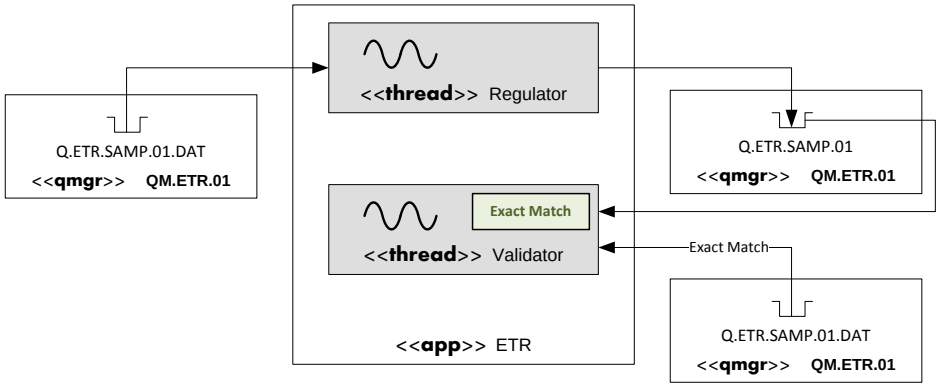


Figure 36 – Send Message to Queue and Validate from the Queue

Test Data is saved in the queue **Q.ETR.SAMP.01.DAT** and transferred to the queue **Q.ETR.SAMP.01** as the Results by the Regulator and compared to the canned results saved in the queue **Q.ETR.SAMP.01.DAT** by the Validator.

```
<TestCase id="ETR_01_2.3_" descr="Exact Match - Queue to File Regulator">
        <Condition>
                - Send Test Data from a queue to a queue
                - Retrieve data from the output queue
                - Validate data against existing data in a queue
        </Condition>
        <Dependencies>
                <Predecessor testCase="ETR_01_2.d_"/>
        </Dependencies>

        <Regulator>
                <TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01.DAT"/></TestData>
                <Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01"/></Results>
                <ExpectedResults>
                        <Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.01.DAT"/>
                </ExpectedResults>
                <RunOutput><File path="" format="raw"/></RunOutput>
        </Validator>
</TestCase>
```

6.1.7. Test Run Sample 01

The following shows an output from the console when the Sample 01 Test Suite is executed.

```
Test Robot started ...
Running Test Suite file
'C:\_Backup\Assets\Assets.dev\assets\ETR\ETR.dev\ETR.Java\samp\ETR.sample\samp_01\unit\ETR_Sam
p_01.testsuite'

Successfully loaded Test Suite configuration.

Test Run Id = ETR_01_1.0.0_UT.00004

Starting Initiators ...
Test Suite successfully initialized.
Test Robot is running TESTCASE(1 of 6) = 'ETR_01_1.1_' ... done [/].
Test Robot is running TESTCASE(2 of 6) = 'ETR_01_1.2_' ... done [/].
Test Robot is running TESTCASE(3 of 6) = 'ETR_01_2.d_' ... done [/].
Test Robot is running TESTCASE(4 of 6) = 'ETR_01_2.1_' ... done [/].
Test Robot is running TESTCASE(5 of 6) = 'ETR_01_2.2_' ... done [/].
Test Robot is running TESTCASE(6 of 6) = 'ETR_01_2.3_' ... done [/].
Starting Janitors ...
Test Suite successfully cleaned.
Test Suite successfully completed.
```

6.2. Sample 02: Supported Headers

The sample demonstrates the use of different headers with Data Source combinations. The supported Header in this release is MQMDHeader.

6.2.1. Test Data = file with MQMD

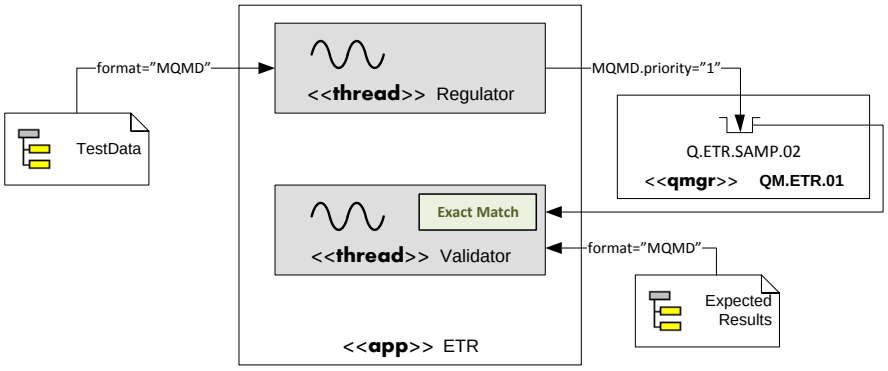


Figure 37 - Read data and MQMD from file and send to queue

Data in the file contains MQMD header parameters that need to be set to the message in the queue. The data file in this sample sets the priority as follows:

```
<MQMD priority="1" />
```

The expected results also contain the MQMD set to priority 1 so that it can be validated by the Validator.

```
<TestCase id="ETR_02_1.1_" descr="Setting header and Validating data and header">
        <Condition>
                - Send Test Data from a file to a queue
                - Sets priority="1"
                - Validate data against existing data in a file
        </Condition>
        <Regulator>
                <TestData><File path="data/ETR_02" format="MQMD"/></TestData>
                <Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/></Results>
                <ExpectedResults><File path="results/ETR_02" format="MQMD"/></ExpectedResults>
                <RunOutput><File path="" format="MQMD"/></RunOutput>
        </Validator>
</TestCase>
```

6.2.2. Test Data = file with MQMD, Failed Validation

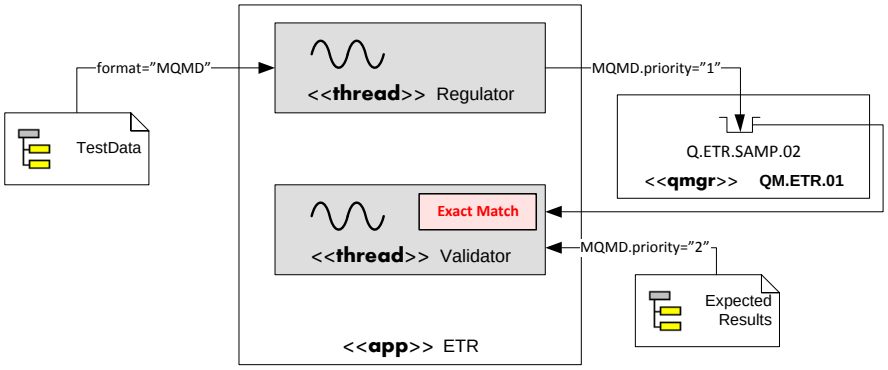


Figure 38 - Read data and MQMD from file and send to queue, wrong header in Expected Results

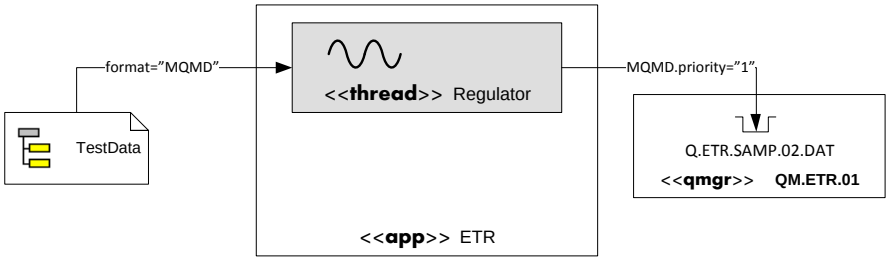
Data in the file contains MQMD header parameters that need to be set to the message in the queue. The data file in this sample sets the priority as follows:

```
<MQMD priority="1" />
```

The expected results also contain the MQMD set to priority 2. Validator checks the values in the header and fails.

```
<TestCase id="ETR_02_1.2x" descr="Setting header and Validating header failure">
	<Condition>
		- Send Test Data from a file to a queue
		- Sets priority="1"
		- Validate data against existing data in a file set priority="2"
	</Condition>
	<Regulator>
		<TestData><File path="data/ETR_02" format="MQMD"/></TestData>
		<Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/></Destination>
	</Regulator>
	<Validator>
		<Latency hr="0" min="0" sec="5"/>
		<Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/></Results>
		<ExpectedResults>
			<File path="results/ETR_02_1.2" format="MQMD"/>
		</ExpectedResults>
		<RunOutput><File path="" format="MQMD"/></RunOutput>
	</Validator>
</TestCase>
```

6.2.3. Load Queue, dependent Test Case



Test Case: ETR_02_2.d_

Figure 39 - Load data with MQMD from file and send to queue

Data in the file contains MQMD header parameters (priority set to 1) that need to be set to the message in the queue.

```
<TestCase id="ETR_02_2.d_" descr="Loads the data queue">
        <Condition>
                - Loads the data queue for running tests with Test Data from queue
        </Condition>
        <Initialize>
                <Clear>
                        <Queue name="Q.ETR.SAMP.02" qmConnRef="ETR.MQ"/>
                        <Queue name="Q.ETR.SAMP.02.DAT" qmConnRef="ETR.MQ"/>
                </Clear>
        </Initialize>
        <Regulator>
                <TestData><File path="data/ETR_02/test01.dat" format="MQMD"/></TestData>
                <Destination><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02.DAT"/></Destination>
        </Regulator>
</TestCase>
```

6.2.4. Test Data = queue, Validation from Queue

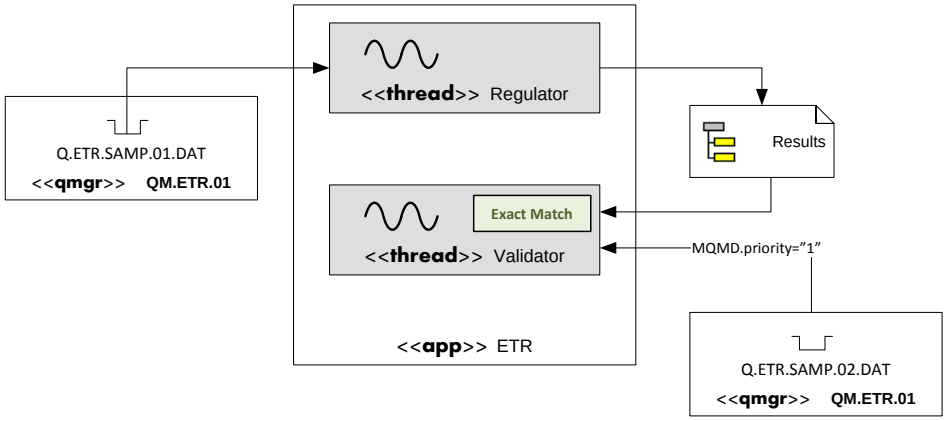


Figure 40 - Read data from queue and send to file

Data in the queue with priority set to 1 send to the file. The Validator looks at the data from the queue and matches against the output Result file.

```
<TestCase id="ETR_02_2.1_" descr="Validator data and MQMDHeader">
        <Condition>
                - Send Test Data from a queue to a file
                - Validate data against existing data in a queue
        </Condition>
        <Dependencies>
                <Predecessor testCase="ETR_02_2.d_"/>
        </Dependencies>

        <Regulator>
                <TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02.DAT"/></TestData>
                <Destination><File path="out/ETR_02_2.1" format="MQMD"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><File path="out/ETR_02_2.1" format="MQMD"/></Results>
                <ExpectedResults>
                     <Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02.DAT"/>
                </ExpectedResults>
                <RunOutput><File path="" format="MQMD"/></RunOutput>
        </Validator>
</TestCase>
```

6.2.5. Test Data = queue, Wrong Header

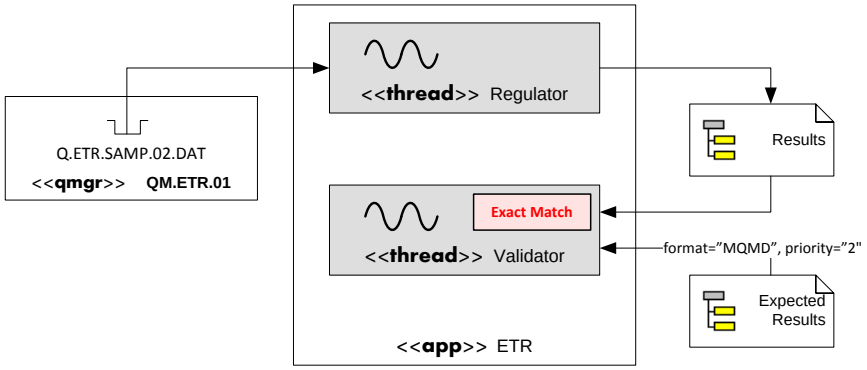


Figure 41 - Read data from queue and send to file, validation of header fails

Data in the queue with priority set to 1 send to the file. The Validator looks at the data from the file (priority set to 2) and fails to match against the output Result file.

```
<TestCase id="ETR_02_2.2x" descr="Validator data and MQMDHeader - fails in MQMD validation">
	<Condition>
		- Send Test Data from a queue to a file
		- Validate data against existing data in a queue
	</Condition>
	<Dependencies>
		<Predecessor testCase="ETR_02_2.d_"/>
	</Dependencies>

	<Regulator>
		<TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02.DAT"/></TestData>
		<Destination><File path="out/ETR_02_2.2" format="MQMD"/></Destination>
	</Regulator>
	<Validator>
		<Latency hr="0" min="0" sec="5"/>
		<Results><File path="out/ETR_02_2.2" format="MQMD"/></Results>
		<ExpectedResults>
			<File path="results/ETR_02_2.2" format="MQMD"/>
		</ExpectedResults>
		<RunOutput><File path="" format="MQMD"/></RunOutput>
	</Validator>
</TestCase>
```

6.2.6. Test Data = queue, Wrong Data

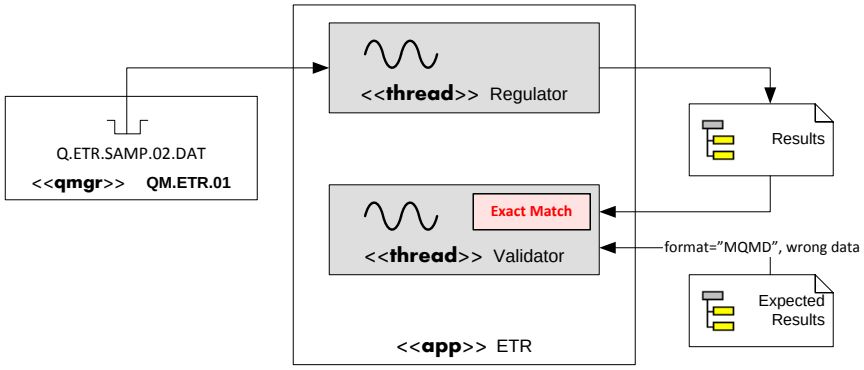


Figure 42 - Read data from queue and send to file, validation of data fails

Data in the queue with priority set to 1 send to the file. The Validator looks at the data from the file (priority set to 1 but wrong data) and fails to match against the output Result file.

```
<TestCase id="ETR_02_2.3x" descr="Validator data and MQMDHeader - fails in Data validation">
        <Condition>
                - Send Test Data from a queue to a file
                - Validate data against existing data in a queue
        </Condition>
        <Dependencies>
                <Predecessor testCase="ETR_02_2.d_"/>
        </Dependencies>

        <Regulator>
                <TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02.DAT"/></TestData>
                <Destination><File path="out/ETR_02_2.3" format="MQMD"/></Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><File path="out/ETR_02_2.3" format="MQMD"/></Results>
                <ExpectedResults>
                        <File path="results/ETR_02_2.3" format="MQMD"/>
                </ExpectedResults>
                <RunOutput><File path="" format="MQMD"/></RunOutput>
        </Validator>
</TestCase>
```

6.2.7. Test Data = queue, Over-riding MQMD

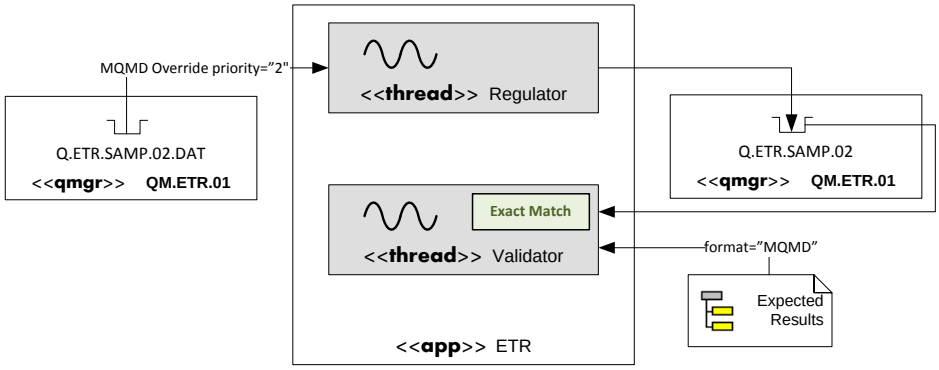


Figure 43 - Read data from queue and send to queue with header override

Data in the queue with priority set to 1 is send to the queue with a header override in the Test Case to set priority to 2. The Validator looks at the data from the file (priority set to 2) and matches against the output Result file.

```
<TestCase id="ETR_02_3.1_" descr="Validator data and MQMDHeader override from Queue">
        <Condition>
                - Send Test Data from a queue to a queue
                - Set the header to a different value
                - Validate data against existing data in a file
        </Condition>
        <Dependencies>
                <Predecessor testCase="ETR_02_2.d_"/>
        </Dependencies>

        <Regulator>
                <TestData><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02.DAT"/></TestData>
                <Destination>
                        <HeaderOverride><MQMDHeader priority="2"/></HeaderOverride>
                        <Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/>
                </Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/></Results>
                <ExpectedResults>
                        <File path="results/ETR_02_3.1" format="MQMD"/>
                </ExpectedResults>
                <RunOutput><File path="" format="MQMD"/></RunOutput>
        </Validator>
</TestCase>
```

6.2.8. Test Data = file, Over-riding MQMD

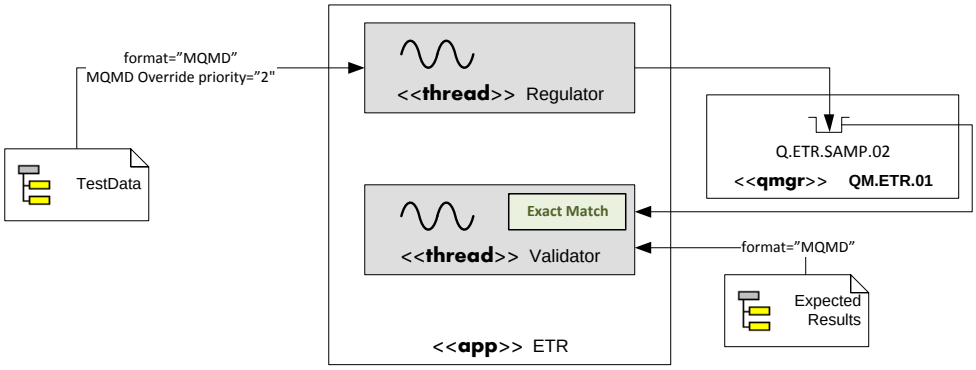


Figure 44 - Read data from file and send to queue with header override

Data in the file with priority set to 1 is send to the queue with a header override in the Test Case to set priority to 2. The Validator looks at the data from the file (priority set to 2) and matches against the output Result file.

```
<TestCase id="ETR_02_3.2_" descr="Validator data and MQMDHeader override from File">
        <Condition>
                - Send Test Data from a queue to a queue
                - Set the header to a different value
                - Validate data against existing data in a file
        </Condition>

        <Regulator>
                <TestData><File path="data/ETR_02" format="MQMD"/></TestData>
                <Destination>
                        <HeaderOverride><MQMDHeader priority="2"/></HeaderOverride>
                        <Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/>
                </Destination>
        </Regulator>
        <Validator>
                <Latency hr="0" min="0" sec="5"/>
                <Results><Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.02"/></Results>
                <ExpectedResults>
                        <File path="results/ETR_02_3.2" format="MQMD"/>
                </ExpectedResults>
                <RunOutput><File path="" format="MQMD"/></RunOutput>
        </Validator>
</TestCase>
```

6.2.9. Test Run Sample 02

The following shows an output from the console when the Sample 02 Test Suite is executed.

```
Test Robot started ...
Running Test Suite file
'C:\_Backup\Assets\Assets.dev\assets\ETR\ETR.dev\ETR.Java\samp\ETR.sample\samp_02\unit\ETR_Sam
p_02.testsuite'

Successfully loaded Test Suite configuration.

Test Run Id = ETR_1.0.0_UT.00073

Starting Initiators ...
Test Suite successfully initialized.
Test Robot is running TESTCASE(1 of 8) = 'ETR_02_1.1_' ... done [/].
Test Robot is running TESTCASE(2 of 8) = 'ETR_02_1.2x' ... done [X].
Test Robot is running TESTCASE(3 of 8) = 'ETR_02_2.d_' ... done [/].
Test Robot is running TESTCASE(4 of 8) = 'ETR_02_2.1_' ... done [/].
Test Robot is running TESTCASE(5 of 8) = 'ETR_02_2.2x' ... done [X].
Test Robot is running TESTCASE(6 of 8) = 'ETR_02_2.3x' ... done [X].
Test Robot is running TESTCASE(7 of 8) = 'ETR_02_3.1_' ... done [/].
Test Robot is running TESTCASE(8 of 8) = 'ETR_02_3.2_' ... done [/].
Starting Janitors ...
Test Suite successfully cleaned.
Test Suite successfully completed.
```

6.3. Sample 99: Performance Test

The sample demonstrates the use of different options while doing performance tests.

6.3.1. Regulated 30 msg/sec into a queue with loop

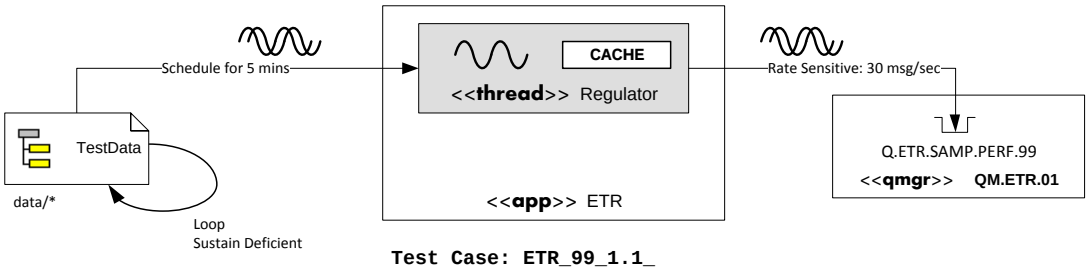


Figure 45 - Rate Regulation Mode

Test Data is saved in the directory **data** and send to the Queue at a specific rate regulated at 30 msg/sec. The autonomic algorithm determines the spawning of Data Loader threads to read forward the files into the cache in a loop due to a small sample data set. Independent Data Sender threads transfer the data from the cache and to the queue destination **Q.ETR.SAMP.PERF.99**.

```
<TestCase id="ETR_99_1.1_" descr="Regulating messages at specific rates for specific
durations.">
        <Condition>
                - Send Test Data from a file to a queue
                - Loop through limited data
                - Sustain the test even if the rate went below the tolerance
        </Condition>
        <Regulator>
                <Schedule><End duration="300"/></Schedule>
                <TestData><File path="data" format="raw"/></TestData>
                <RegulationMode>
                        <RateSensitive msgRate="30" loop="true" sustainDeficiency="-1"/>
                </RegulationMode>
                <Destination inherritTR="true">
                        <Queue qmConnRef="ETR.MQ" name="Q.ETR.SAMP.PERF.99"/>
                </Destination>
        </Regulator>
</TestCase>
```

6.3.2. Test Run Sample 99

The following shows an output from the console when the Sample 99 Test Suite is executed.

```
Test Robot started ...
Running Test Suite file
'C:\_Backup\Assets\Assets.dev\assets\ETR\ETR.dev\ETR.Java\samp\ETR.sample\samp_99\perf\ETR_Sam
p_99.testsuite'

Successfully loaded Test Suite configuration.

Test Run Id = ETR_01_1.0.0_PT.00006

Starting Initiators ...
Test Suite successfully initialized.
Test Robot is running TESTCASE(1 of 1) = 'ETR_99_1.1_' ...
.............|.............|.............|.............| (00:01:00)
.............|.............|.............|.............| (00:02:00)
.............|.............|.............|.............| (00:03:00)
.............|.............|.............|.............| (00:04:00)
.............|.............|.............|.............| (00:05:00)
..
Target Rate = 30 msg/sec
Average Rate achieved = 32.85179381239145 msg/sec
No. of Messages Processed = 9900
Total Processing = 301.353407261 sec
 done [/].
Starting Janitors ...
Test Suite successfully cleaned.
Test Suite successfully completed.
```

7. Test Scenario

In this section more complex real-life Test Cases are documented for the purpose of demonstration of capabilities as well as guidelines for project specific Test Cases that needs to be created.

7.1. Scenario 1: Train Tracker

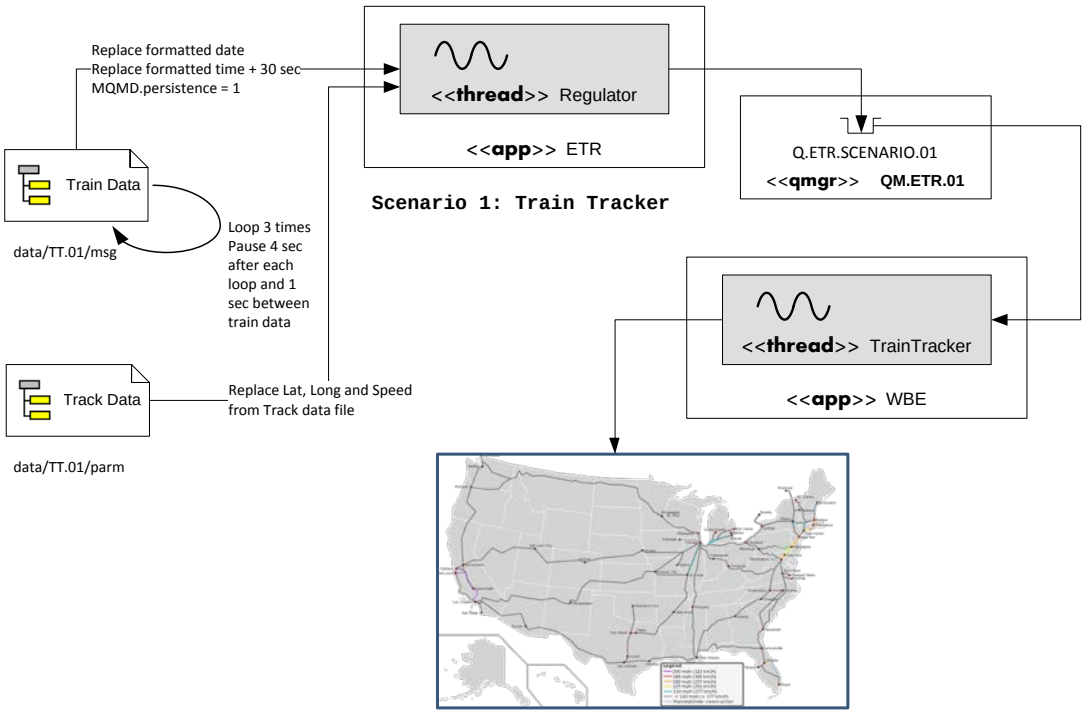


Figure 46 - Train Tracker

Problem Description:

An event correlating application captures events generated by the trains at regular intervals. The events are sent through the cellular network or other available networks to a correlation hub with various data points related to the train. The application also checks if the specific train follows its track based on the GPS coordinates and rejects data that does not follow the track. The application plots the train on the map at real-time. The objective is to send data to the application for testing multiple trains each on its own track and validate if the application plots the correct information on the map. Validation is visual inspection.

Testing Strategy:

Read Train data from a file and replace marked variables from a parameter file. Each train is represented as a data file and its corresponding parameter file. The parameter file contains the latitude, longitude and speed. For each interval the data is read from the file and replaced from the parameter file and send to the queue.

Configuration:

For the sake of simplicity only 2 trains are considered here – **Train 001** and **Train 002**. The data is stored in the **data/TT.01/msg** and the corresponding parameter files in **data/TT.01/parm**. Sample time is also replaced by the formatted current timestamp. Storage time is replaced as 30 sec added to the current timestamp and formatted. Since the sample only contains 4 rows in each parameter file, the data loop is set to 3 times. A wait interval of 1 second will be considered. The interval at which data will be simulated is 5 sec. So the wait to be set is 4 sec.

```
<TestCase id="TT.01" descr="Sending WiTronix Feed">
    <Condition>
        Sending WiTronix Train information to the TT application queue
        Transforming:
            WSampleTime - with current datetime
            WStorageTime       - with current datetime +30 sec
            WLocoLatitude      - with track info file latitude
            WLocoLongitude     - with track info file longitude
            WLocoSpeed  - with track info file speed
    </Condition>
    <Regulator>
        <TestData><File path="data/TT.01/msg" format="raw"/></TestData>

        <TransformationRule replace="$DateFormat('MM-dd-yyyy hh:mm:ss')."> 
            <Literal string="*WSampleTime*"/>
        </TransformationRule>
        <TransformationRule replace="$DateChange('+30s','MM-dd-yyyy hh:mm:ss')."> 
            <Literal string="*WStorageTime*"/>
        </TransformationRule>
        <TransformationRule    replace="$[WLOCOLAT]"
                               keywordParms="WLOCOLAT,WLOCOLONG,WLOCOSPEED"
                               keywordFile="data/TT.01/parm">
            <Literal string="*WLocoLatitude*"/>
        </TransformationRule>
        <TransformationRule    replace="$[WLOCOLONG]"
                               keywordParms="WLOCOLAT,WLOCOLONG,WLOCOSPEED"
                               keywordFile="data/TT.01/parm">
            <Literal string="*WLocoLongitude*"/>
        </TransformationRule>
        <TransformationRule    replace="$[WLOCOSPEED]"
                               keywordParms="WLOCOLAT,WLOCOLONG,WLOCOSPEED"
                               keywordFile="data/TT.01/parm">
            <Literal string="*WLocoSpeed*"/>
        </TransformationRule>

        <RegulationMode>
            <PauseProcessor dataPause="1" loop="3" loopPause="4"/>
        </RegulationMode>

        <Destination inherritTR="true">
            <HeaderOverride>
                <MQMDHeader persistence="1" format="MQSTR   "/>
            </HeaderOverride>
            <Queue qmConnRef="MQ.TT" name="Q.ETR.SCENARIO.01"/>
        </Destination>
    </Regulator>
</TestCase>
```

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

2. OMG 11-08-06 – UML 2.4.1 Superstructure specification
   http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/

3. The Unified Modeling Language Reference Manual by James Rambaugh, Ivar Jacobson and Grady Booch
   http://www.dcc.fc.up.pt/~zp/aulas/1314/asw/geral/bibliografia/Addison%20Wesley%20-%20UML%20Reference%20Manual.pdf

4. UML 2.5 Visio Stencil
   http://www.softwarestencils.com/uml/index.html

5. WMQ 7.5 Knowledge Center
   http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/welcome/WelcomePagev7r5.html?lang=en

6. IIB 9.0 Knowledge Center
   http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.msgbroker.helphome.doc/help_home_msgbroker.htm?lang=en

7. Performance Harness for Java Message Service
   https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=1c020fe8-4efb-4d70-afb7-0f561120c2aa

8. IBM Connectivity, Integration and SOA
   http://www-03.ibm.com/software/products/en/category/connectivity-integration-soa

9. nmon – Performance Analysis Tool by Neigel
   http://www.ibm.com/developerworks/aix/library/au-analyze_aix/

10. 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

11. Books:

    - WebSphere Message Broker Basics
      http://www.redbooks.ibm.com/redbooks/pdfs/sg247137.pdf
    - Managing WebSphere Message Broker Resources in a Production Environment
      http://www.redbooks.ibm.com/redbooks/pdfs/sg247283.pdf
    - WebSphere MQ Queue Manager Clusters
      http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC34658900