



SOA Testing Framework Developer Guide

|  |  |
| --- | --- |
| Author: | Ladislav Jech |
| Owner: | Irish Water Programme |
| Document Date: |  |
| Document Reference: |  |
| Version: | 0.1 |

Document History

Document Location

This is a snapshot of an on-line document. Paper copies are valid only on the day they are printed. Refer to the author if you are in any doubt about the currency of this document.

Most up-to-date version is available at SVN repository:

<http://10.19.12.41:8080/svn/repos_int1/m2/trunk/soa_testing_framework/doc/user-guide>

Revision History

|  |  |
| --- | --- |
| Date of this revision: 04.11.2013 | Date of next revision *31.12.2013* |

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Revision Date | Summary of Changes | Changes marked |
| 0.1 | 04.11.2013 | Initial version |  |
|  |  |  |  |
|  |  |  |  |

Approvals

This document requires following approvals:

|  |  |
| --- | --- |
| Name | Title |
| Kieran Murphy | Integration Architect |

Distribution

This document has been distributed to

|  | Title | Name | Title |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

1. Introduction 4

2. High Level Overview 5

2.1 High Level Framework Design 5

2.2 High Level Framework Process Flow 5

2.3 SOA Testing Framework within existing environment 5

2.4 SOA Testing Framework within future environment 5

2.5 IBM Programme Implementation Integration Patterns 5

2.5.1 OSB Patterns 5

2.5.2 SOA Patterns 6

3. Detail Process Flow 7

3.1 Pattern: Database -> Queue -> Queue -> Database/SOAP 7

3.2 Pattern: CSV File -> Queue -> Database/SOAP 7

4. General JAVA code guidelines 8

4.1 Object OperationResult 8

4.2 Logging 9

4.3 Exception handling 9

# Introduction

SOA Testing Framework project has been created within the Irish Water Programme as an evolutional mechanism requirement which should help to transform the software development life cycle to the way of continuous integration and delivery, both becoming standards in those days. Framework itself provides mechanisms to test defined interfaces and create testing reports.

The biggest purpose is probably to save time of developers/testers on interface testing phases.

Framework is fully implemented using Java Standard Edition 7 and supports XML based configuration with 2 runtime access modes:

* Graphical User Interface

Used from developer/tester private laptop to provide user friendly control interface

* Command Line

Used by Hudson server for automatic execution of tests

# High Level Overview

This chapter describes high level process flow and design of the framework.

## High Level Framework Design

The framework itself is designed in a way to support endpoints based on multiple technologies. For each technology type there is predefined set of integration operations related strictly to one technology type. Currently supported are following technologies and operations:

Figure - High Level Framework Design

The framework is designed with idea of future usage, but the main criteria in the moment is to use the product ASAP, this fall into 2 designs of the framework flow usages:

1. SOA Testing Framework within existing environment
2. SOA Testing Framework within future continuous integration environment

Both of those ways of usage are more described bellow in related chapters.

## High Level Framework Process Flow

asdfa

## SOA Testing Framework within existing environment

asdfdsafsa

## SOA Testing Framework within future environment

## IBM Programme Implementation Integration Patterns

### OSB Patterns

There are few of integration patterns identified within IBM Irish Water Programme implementation which are currently supported. Here they are:

1. Database -> Queue -> Database/SOAP

### SOA Patterns

# Detail Process Flow

In this chapter I am going to describe detailed step-by-step process of the configuration and execution of the framework.

## Pattern: Database -> Queue -> Queue -> Database/SOAP

## Pattern: CSV File -> Queue -> Database/SOAP

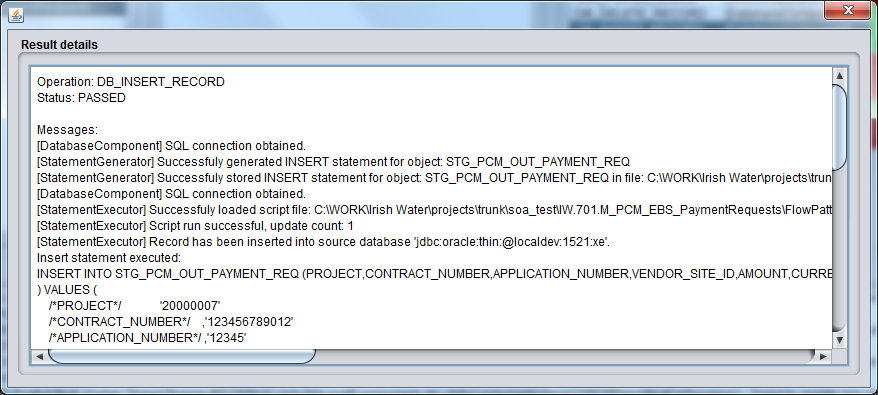
# General JAVA code guidelines

## Object OperationResult

Serves as an indicator of success or failure at the operation's execution end and also holds a list of all messages gathered during the operation's execution. The purpose of the success/failure indicator is obvious - you can tell whether the operation was successful or not (such as if the DB operation DB\_INSERT\_RECORD succeeded at inserting a record to the database). The list of messages is the main source of the data that goes to the test report and also, in case of the GUI, it is used to display what was going on during the operation's execution in a step-by-step manner (check the screenshot below).

Current instance of this class is accessible anywhere in the code simply by calling a static method OperationResult.getInstance(). This instance lives from the very beginning of the operation's execution to its very end. FlowExecutor class takes care of "resetting" it at each operation execution start. No other class outside of com.ibm.soatf.flow package can reset the state of it. By resetting we mean creating entirely new instance of OperationResult, effectively setting the overall success to false and clearing any messages it contained prior to reset.

public void addMsg(String msg):

* use it everywhere you want to record a certain event which will then be displayed in the operation result details window (after double-clicking the row in the results table in GUI)
* each added message automatically contains a name of the class from which you called the method (in brackets): 
* it is considered a good practice to use OperationResult.addMsg() wherever you also log message using the framework's logging mechanism but not necessarily everywhere. It's entirely up to you what you want to appear in the details (maybe you want the log to contain a TRACE message of opening/closing a database connection but it's not really necessary in the report or details window)
* always use it in the place where you are catching a non-framework exception (such as SQLException, IOException, ...), wrapping it in FrameworkException (or its subclass) and rethrowing it. Try to also include the original exception's message
* similarly, use it wherever you are creating and throwing a brand new FrameworkException

public boolean isSuccessful():

* tells if operation's execution ended in success (true) or failure (false)

public void markSuccessful():

* this method sets the operation's execution result to success
* you call this method at the point where you consider the operation succeeded

NOTE: markSuccessful() method introduces a possible danger: once it is called, there is no possibility to change the operation's result back to failure anymore so you should be really careful to call it at the very end of the operation

## Logging

Framework uses Log4j2 as its logging facility so standard logging methods are used. Each class that uses logging should instantiate its own Logger instance at the class body beginning, for example:

public class DatabaseComponent **extends** SOATFComponent **{**

private static final Logger logger **=** LogManager**.**getLogger**(**DatabaseComponent**.**class**);**

**...**

* when you log a message via logger, always consider to call also the OperationResult.addMsg() at the same time
* DO NOT use logger in the catch blocks unless you don't propagate the exception further - this exception is always logged at the end of the operation's execution automatically

## Exception handling

The main exception class is FrameworkException. It has 2 subclasses: FrameworkExecutionException and FrameworkConfigurationException. All these exceptions are checked exceptions.

Each checked exception that is not FrameworkException (or its subclass) - that is, each checked exception thrown by the standard JDK or 3rd party libraries - should be caught and handled appropriately. Unless there are special circumstances you should always rethrow this exception wrapped in one of the framework's exceptions so it eventually propagates to the highest level in the execution chain, FlowExecutor.execute() method. This method is handled automatically and if there was exception thrown by it, it will be logged in the log.

In this example you see SQLException and IOException being handled. Key points here are:

* catch block where message is recorded in the OperationResult object
* new FrameworkExecutionException wrapping the original is created and rethrown
* the method is declared to be throwing FrameworkExecutionException

public static void generateInsertStatement**(**Connection conn**,** DatabaseComponent**.**DbObjectConfig config**,** File file**)** **throws** FrameworkExecutionException **{**

OperationResult cor **=** OperationResult**.**getInstance**();**

**try** **{**

//...

//some logic is here...

//...

String msg **=** "Successfuly stored INSERT statement for object: " **+** objectName **+** " in file: " **+** outputScriptFilePath**;**

logger**.**debug**(**msg**);**

cor**.**addMsg**(**msg**);**

cor**.**markSuccessful**();**

**}** **catch** **(**SQLException ex**)** **{**

String msg **=** String**.**format**(**"Failed to generate INSERT statement. Reason: %s"**,** ex**.**getMessage**());**

cor**.**addMsg**(**msg**);**

**throw** **new** FrameworkExecutionException**(**msg**,** ex**);**

**}** **catch** **(**IOException ex**)** **{**

String msg **=** String**.**format**(**"Failed to save INSERT statement file %s. Reason: %s"**,** outputScriptFilePath**,** ex**.**getMessage**());**

cor**.**addMsg**(**msg**);**

**throw** **new** FrameworkExecutionException**(**msg**,** ex**);**

**}**

**}**