

YAWL Interface X – Description and Developers Guide

Document Control

Date	Author	Version	Change
20 Sept 2006	Michael Adams	1.0	Initial Draft

Preface

This document provides a technical overview of the structure and use of YAWL *Interface X* – an interface through which is passed certain notifications at milestones during the execution of a process instance where a process exception may have occurred or may be tested for.

Introduction

The YAWL system provides support for interconnecting external applications with the workflow execution engine using a service-oriented approach. This enables running workflow instances and external applications to interact with each other in order to delegate work, to signal the creation of process instances and workitems, or to notify a certain events or changes in the status of existing workitems.

Such external applications are referred to as *YAWL Custom Services*. These services interact with the YAWL engine across a number of interfaces designed for particular purposes, supporting the ability to send and receive messages and XML data to and from the engine.

One such interface is *Interface X*, which has been designed to allow the engine to notify custom services of certain milestone events during the execution of a process instance where process exceptions either may have occurred or may be tested for. Thus *Interface X* provides for a custom service to be designed that dynamically captures and handles process exceptions.

An example of a service which does just that is the *Worklet Exception Service*, which is distributed with YAWL Beta 8 (and later) release. In fact, *Interface X* was created to enable the *Worklet Exception Service* to be built, and that service may be viewed as an example of the kinds of things that can be achieved using the interface. However, one of the overriding design objectives was that the interface should be structured for ‘generic application’ – that is, it can be applied by a variety of services that can make use of milestone notifications during process executions.

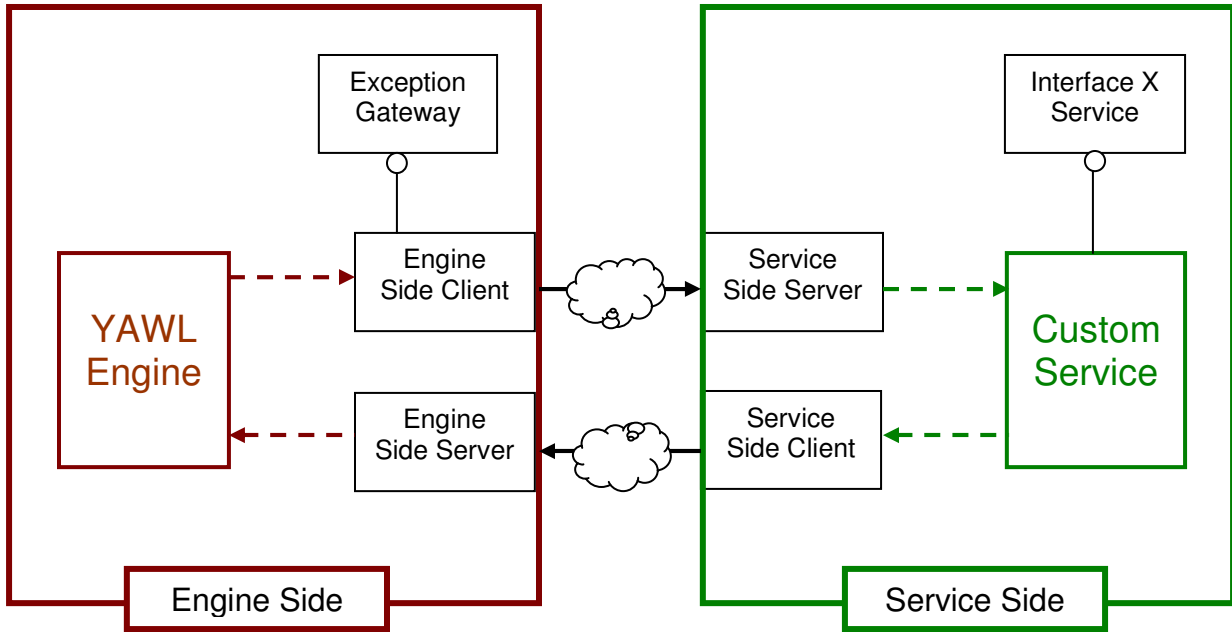
Since it only makes sense to have one custom service acting as an exception handling service at any one time, services that implement *Interface X* have two distinct states – enabled and disabled. When enabled, the engine generates notifications for **every** process instance it executes – that is, the engine makes no decisions about whether a particular process should generate the notifications or not. Thus it is up to the designer of the custom service to determine how best to deal with the notifications. When the service is disabled, the engine generates **no** notifications across the interface. Enabling and disabling an Interface X custom service is achieved via the setting of parameters in the engines *web.xml* configuration file.

The Interface X Package

The package `au.edu.qut.yawl.engine.interface.interfaceX` contains the following entities:

- § *ExceptionGateway*: a java interface that defines the event notifications that pass from the engine side to the custom service side.
- § *InterfaceX_Service*: a java interface that defines the methods that must be implemented by the custom service to receive event notifications.
- § *InterfaceX_EngineSideClient*: implements the *ExceptionGateway* interface. Receives the event notifications from the engine, translates them and their data sets to POST messages and passes them across the interface.
- § *InterfaceX_ServiceSideServer*: a java Servlet that receives the POST messages passed across the interface from the *EngineSideClient*, and translates them and their data sets into method calls to the custom service (i.e. the methods implemented from the *InterfaceX_Service* interface).
- § *InterfaceX_ServiceSideClient*: has a number of methods that are called from the custom service. Translates the method calls and their datasets into POST messages and passes them across the interface.
- § *InterfaceX_EngineSideServer*: a java Servlet that receives the POST messages passed across the interface from the *ServiceSideClient*, and translates them and their data sets into method calls to the engine. Also registers the custom service with the engine via parameters in the engine's *web.xml* configuration file (see *Configuring an Interface X Custom Service* below).

The logical layout of the interface can be seen schematically below:



The items of interest to a custom service developer are those on the service side. These are described in some detail below.

InterfaceX_Service

This java interface defines eight methods that must be implemented by a custom service wishing to access *Interface X*, seven of which are notification announcements from the engine side:

§ `handleCheckCaseConstraintEvent(String specID, String caseID, String data, boolean preCheck)`

This method call is invoked by the engine side at two points in the lifecycle of a case (i.e. process instance), firstly when the case is launched and again when it completes. It is designed to allow a custom service to check the case's starting and concluding data to ensure they are valid, and to take appropriate action as required but of course can be used for other purposes.

- `specID` is the name of the specification for the case
- `caseID` is the identification number of the case instance
- `data` is a String of XML formatted data containing relevant case information
- `preCheck` is true to denote the case has been launched, false to denote its completion.

§ **handleCheckWorkItemConstraintEvent(WorkItemRecord wir,
String data, boolean preCheck)**

This method call is invoked by the engine side at two points in the lifecycle of a workitem, firstly when the workitem is enabled and again when it completes. It is designed to allow a custom service to check the workitem's data to ensure it is valid, and to take appropriate action as required before and after the workitem is executed, but of course can be used for other purposes.

- `wir` is the workitem record of the item in question
- `data` is a String of XML formatted data containing relevant workitem information
- `preCheck` is true to denote the workitem is enabled, false to denote its completion.

§ **handleTimeoutEvent(WorkItemRecord wir, String taskList)**

This method call is invoked by the engine when a timed workitem (that is, a workitem that has been associated with the YAWL Time Service) reaches its deadline.

- `wir` is the workitem record of the timed item in question
- `tasklist` is a concatenated String of task identifiers of the form “[taskA, taskB, taskX]” representing a list of tasks which had workitems running in parallel to the timed workitem when the timeout occurred.

§ **handleCaseCancellationEvent(String caseID)**

This method call is invoked by the engine when a case is cancelled.

- `caseID` is the identification number of the case instance

§ **handleResourceUnavailableException(WorkItemRecord wir)
handleWorkItemAbortException(WorkItemRecord wir)
handleConstraintViolationException(WorkItemRecord wir)**

These three methods are included in the interface for future implementation – that is, they are not yet implemented on the engine side and so are never called. However, since they are included in the interface, any custom service that implements it must include these definitions as empty methods.

§ **doGet(HttpServletRequest request,
HttpServletResponse response)**

Handles the servlet request that occurs when a browser client navigates to the custom service's URI. Should be used to display a web page.

InterfaceX_ServiceSideClient

The Service-Side Client class provides a number of methods that may be called from the custom service and invoked on the engine side. The list below provides a brief overview of each. Most require an active admin session with the engine (a `sessionHandle`) - see the documents *Interface B Summary* and *CustomWebServices* for details on connecting to the engine with an active session.

```
String setExceptionObserver(String observerURI)  
                               throws IOException
```

Purpose: Registers the URI of the custom service with the engine. If successful, enables the custom service, which starts the notification of events across the interface.

Arguments: `observerURI` – the full URI of the custom service (e.g. “http://localhost:8080/myCustomService”)

Returns: a `String` denoting success or failure.

.....
String removeExceptionObserver() throws IOException

Purpose: Removes the URI of the custom service from the engine, effectively disabling the custom service.

Arguments: `nil`

Returns: a `String` denoting success or failure.

.....
**void updateWorkItemData(WorkItemRecord wir, Element data,
 String sessionHandle) throws IOException**

Purpose: Maps the data values passed to the workitem specified on the engine side. Used to update the values of a workitem's data attributes, stored in the engine, with the values of identically named attributes in the `Element` passed.

Arguments: `wir` – the workitem to update
`data` – a `JDOM Element` containing data attributes and values

sessionHandle – the session handle of an active admin session

Returns: nil

.....
**void updateCaseData(String caseID, Element data,
String sessionHandle) throws IOException**

Purpose: Maps the case (or net) level data values passed to the case specified on the engine side. Used to update the values of case-level data attributes, stored in the engine, with the values of identically named attributes in the Element passed.

Arguments: caseID – the identifier of the case to update
data – a JDOM Element containing data attributes and values
sessionHandle – the session handle of an active admin session

Returns: nil

.....
**void forceCompleteWorkItem(WorkItemRecord wir, Element data,
String sessionHandle) throws IOException**

Purpose: Force-completes the specified workitem. A force-completed workitem receives a final status of “ForcedComplete” and the process continues to the next task.

Arguments: wir – the workitem to force-complete
data – a JDOM Element containing the workitem’s final data values
sessionHandle – the session handle of an active admin session

Returns: nil

.....
**WorkItemRecord unsuspendWorkItem(String workItemID,
String sessionHandle) throws IOException**

Purpose: Continues (i.e. unsuspends) the specified workitem. Does nothing if the workitem is not already suspended.

Arguments: workItemID – the identifier of the workitem to continue
sessionHandle – the session handle of an active admin session

Returns: A WorkItemRecord referring to the updated workitem

.....

```
void restartWorkItem(String workItemID, String sessionHandle)
                        throws IOException
```

Purpose: Restarts the specified workitem. Sets its status as “Enabled” and resets the items data values to those when it was initialised.

Arguments: workItemID – the identifier of the workitem to restart
sessionHandle – the session handle of an active admin session

Returns: nil

.....

```
void startWorkItem(String workItemID, String sessionHandle)
                    throws IOException
```

Purpose: Starts the specified workitem in the engine.

Arguments: workItemID – the identifier of the workitem to start
sessionHandle – the session handle of an active admin session

Returns: nil

.....

```
void cancelWorkItem(String workItemID, String sessionHandle)
                    throws IOException
```

Purpose: Cancels the specified workitem. Sets its status as “Failed” - no further processing is enabled on the same branch as the failed workitem.

Arguments: workItemID – the identifier of the workitem to cancel
sessionHandle – the session handle of an active admin session

Returns: nil

Configuring an Interface X Custom Service

There are two important configuration tasks that must be carried out before a custom service that implements *Interface X* will function.

1. In your custom service, create a variable of type `InterfaceX_ServiceSideClient` and pass to its constructor the URI of the YAWL engine, with the suffix “/ix”. For example, in the constructor of your custom service:

```
private InterfaceX_ServiceSideClient ixClient = new
    InterfaceX_ServiceSideClient("http://localhost:8080/yawl/ix");
```

2. In the engine’s `web.xml` file (found in your Tomcat installation’s “`webapps/yawl/WEB-INF`” directory), find the parameter called *ExceptionObserverURI* and change it to the URI of your custom service, for example:

```
<context-param>
  <param-name>ExceptionObserverURI</param-name>
  <param-value>http://localhost:8080/myCustomService/ix</param-value>
  <description>
    This value provides the URI of an Exception Service that
    monitors for process exceptions through Interface X.
  </description>
</context-param>
```

While you are in the engine’s *web.xml*, you may want to also enable the custom exception service (in the *EnableExceptionService* parameter)

As mentioned previously, the Worklet Exception Service makes extensive use of *Interface X* and so is a good example to browse for tips and pointers.