Working with Task Workflows

June 2012

webMethods

Working with Task Workflows

Applicability

The information in this article applies to the task workflow/form flow capability included with MWS_8.2_SP1_Fix9 - released on 26 June 2012.

My webMethods Server 8.2.1 with Fix 9 installed is required. You can obtain FIx 9 from the Empower Web site (login required) or with the Software AG Update Manager.

Task Workflow Overview

Tasks are closely integrated with webMethods business process models. A task can be added to a business process as a *user task activity*, also referred to as a *user task*. Each user task can represent a discrete human activity within the process, such as installing a piece of equipment, but in more advanced processes, you might want to create a series of user tasks, where each user task step represents one activity in a larger, overall procedure.

When you connect a series of tasks together in this way, you are creating a *task workflow*, also referred to as a *form flow*.

A task workflow enables you to break up a complicated procedure into a series of simpler user task steps. Task orchestration is done by the process model and not inside a single task that could become overly complex and difficult to deploy, maintain, use, and understand.

In a task workflow, you can use the more sophisticated logic of the process model to determine which task to start and the next task interface to display to the user. The process model can call additional services to implement any necessary business logic between tasks steps. As the task workflow progresses, the individual tasks open automatically; the user does not have to manually open each individual task.

In a typical task flow scenario, a previous step in a business process completes and then transitions to and starts (or *queues*) a user task as the next step in the process.

For example:

- Task1 presents an interface that enables the user to gather user personal information from an applicant to run a credit check.
- When the user completes this task, the task submits all the collected data back to the process.
- The process executes some business logic (most likely in the form of one or more services) to determine the applicant's credit score based on the submitted data.
- Depending on the result of the credit check, the process transitions to either a loan application task or to a "credit check failed" report task. The process presents the interface of the next task to the user automatically.

Basic Operation

From the viewpoint of the user, the task workflow appears as a seamless flow of task interfaces in My webMethods. As the user completes each task in the workflow, the next logical interface appears, thus eliminating the need for the user to locate each new task in My Inbox and then open it manually.

Operationally, this behavior is implemented with the following:

- A process component that waits for the next task of the business process (or for completion of the workflow). Upon notification, the waiting component then executes a user interface action, such as open the next task in the workflow, or transition to the next post-workflow activity in the process. The waiting component uses a Java API to wait for the next action to occur.
- A process step that notifies the waiting component. Within the workflow, the next workflow task uses a Java API to notify the waiting step. When the final step in the workflow completes, it waits for notification from a following step that invokes an Integration Server flow service to provide the notification. In both cases, the waiting step can then transition to the next activity.

Code Samples

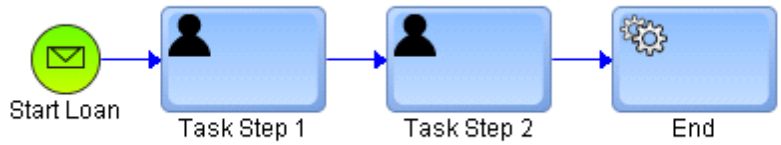
To assist you with understanding how to implement a task workflow, you can examine and deploy a sample task application, process model, and Integration Server package that support a very simple loan application process. You can find the code samples in on the Software AG Community Web site at:

http://communities.softwareag.com/ecosystem/communities/codesamples/webmethods/caf/webmethods-caf-codesamples.html

The sample zip files contain the following items:

- FormFlowProcessTasks.zip. This file contains the FormFlowProcess Tasks task application that contains two tasks, each with the following portlets:
  - StartLoanProcess. The primary purpose of this portlet is to provide a My webMethods interface that enables the user to enter a loan process number and start a new loan process instance. The portlet transitions to the first task interface when the first task is instantiated by the process instance.
  - TaskStep1Overview. This is a standard Task Overview portlet that serves as a container for providing a formatted message panel for displaying any JSF context messages, as well as portlet include controls for any additional portlets in the task.
  - TaskStep1Start. A standard Start portlet that enables a user to start a new task on the Task Engine Administration page of My webMethods. If a task is used exclusively within a process (where it will be triggered only by the process), you do not need to include a task start portlet.

  - TaskStep1View. Also a standard portlet, this is identified as the Task Details portlet in the task editor. It presents editable and non-editable task information, as well as various controls and tabs. This is the portlet where you implement transition to the next task in the task workflow when the task is completed.
- WmFormFlowTest.zip. This file contains the Integration Server package with the required IS document "Loan" as well as the following:
  - startLoanProcess flow service to start loan application process.
  - formFlowComplete flow service to complete a task workflow when process ends.
- FormFlowTest.zip. This file contains the FormFlowProcess process model, a very simple process model that demonstrates a task workflow:



The start message step is subscribed to the WmFormFlowTest IS document named Loan. Task Step 1 and Task Step 2 are examples of a two-stage task workflow, and the process completes with a service activity that calls a notification service.

Implementation Components

You implement a task workflow using these installed software components:

- A Java API com.webmethods.portal.service.task.ITaskFormFlowService included with Task Engine. The Java docs for the Java interface are included with the code samples.
- A built-in Integration Server public service included with the WmTaskClient package pub.task.taskclient:formFlowTaskNotify. For more information about this task service, see the PDF included with the code samples.

Both of these publications are available in the webMethods Product Suite section of the Software AG Documentation Web site.

Working with the Sample Applications

Before you can work with the sample applications you must have My webMethods Server, Integration Server, and Software AG Designer running, and you must:

- Import the process model into Software AG Designer and then build and upload the process to the Integration Server.
- Import the task application into Designer and then publish it to My webMethods Server.

- Unzip the WmFormFlowTest package, add it to the /packages directory of your Integration Server instance, and then activate the package with the Integration Server Administrator.
- Log in to My webMethods Server and access the StartLoanProcess portlet with the following URL: [*hostname*]:[*port*]/start.loan.process. For example: localhost:8585/start.loan.process.

At this point, you can demonstrate a very simple usage scenario with the installed applications:

1 Type a value into the Loan Number field in the Start Loan Process 2 window.

2 Click Start Single Process.

   - An instance of the FormFlowProcess business process starts, and immediately starts Task Step 1, while the StartLoanProcess portlet waits.
   - When the StartLoanProcess portlet is notified that the user task is fully instantiated, the StartLoanProcess portlet redirects to the user task interface URL, which opens the task interface for the end user to interact with.
   - In this sample, Task Step 1 has no functionality other than to display the Loan Number entered in the StartLoanProcess portlet.

3 Click Complete.

   - The process instance starts the next user task in the sequence, Task Step 2, while Task Step 1 waits.
   - When Task Step 1 is notified that the Task Step 2 is fully instantiated, Task Step 1 redirects to the URL of the Task Step 2 interface, and the second task interface opens.
   - Again, Task Step 2 has no functionality other than to display the Loan Number entered in the StartLoanProcess portlet.

4 Click Complete.

   This completes the task workflow, and the process instance transitions to the End service task:

   - The service activity is configured to run the IS service formFlowComplete in the WmFormFlowTest package.
   - The formFlowComplete service contains the WmTaskClient service pub.task.taskclient:formFlowTaskNotify, which enables you to pass a correlation ID, a result, as well as the Loan document.
   - This step can also be configured to provide any required error handling
   - With the completion of the End service activity, the original StartLoanProcess portlet interface appears in My webMethods, and the entire process can begin again. However, if necessary, the business process could continue on from the service step to additional downstream steps in the process.

This workflow can be extended with as many user tasks as necessary. In addition, the logic capabilities of both the process instance and the user task steps can be implemented to provide more advanced handling. For example:

- You can implement task events, task assignment, and the task control set functionality when you design the tasks to apply a certain level of conditional behavior within the tasks applications.
- The task activity steps themselves can be configured with process behavior such as transition looping, join logic, and If Condition transitions for further flexibility. For more information, see the *webMethods BPM Process Development Help* in Software AG Designer.
- Using the available process step logic, you can add service activity steps to the process model to provide any process work that is needed to support the task workflow.

Key Implementation Points

As you examine the sample implementation, make note of the following points:

Single path processing only

Software AG Designer enables you to create parallel data flows in a process model through the use of gateways and step joins and transitions. However, it is not possible to use a task workflow with parallel process branches where each branch might queue its own task instance. That is, you must avoid creating a situation where more than one task instance is running in parallel.

Task workflow correlation ID

A primary concept with the task workflow is that each object in the workflow must wait for the next task to fully instantiate before a transition occurs. This requires you to add a wait mechanism to each component in the workflow. You use a task workflow correlation ID to synchronize the waiting process component and the new task being queued.

This will ensure correct data flow through the process. The value of the task workflow correlation ID must be unique within the Process Engine environment.

Important! The task workflow correlation ID is completely different from and unrelated to the standard document correlation ID often used in process implementation.

Prepare to wait

The first requirement in implementing the wait mechanism is to call:

```
TaskHelper.getTaskFormFlowService().waitPrepare(correlationID)
```

This returns a wait object to be used later to actually wait for notification with the correlation ID. It is very important to call `waitPrepare()` *before* starting a process or completing a task. Because these activities are executed in parallel, there is chance that the notification could be published *before* the wait is started and would then be missed.

Note: When the very first activity step in your process model is a workflow task, you must implement the wait mechanism in the object that starts the process. For example, in the sample StartLoanProcess portlet, when the user clicks the Start Loan Process button, the first activity is to invoke `waitPrepare`. Then, the portlet starts the loan process. Example code can be found at:

```
FormFlowProcessTasks\src\com\webmethods\caf\startloanprocess\
  StartLoanProcessDefaultviewView.java
```

Waiting for task instantiation

The next part of the code is to wait for the new task being queued:

```
TaskHelper.getTaskFormFlowService().wait(waitObject, timeoutinmillis)
```

Notification

After the task is fully instantiated by the Process Engine, the task must notify any objects that are waiting for it. This is accomplished by adding a Queued event to the task and implementing an Invoke Service task action for the event that calls the following API:

```
TaskHelper.getTaskFormFlowService().notify(correlationID, result, false)
```

where `result` can be any value to be passed back to the waiting component. In the example, the result passes the taskURL of new task being queued. The code be found at:

```
FormFlowProcessTasks\src\com\webmethods\caf\taskclient\TaskStep1RuleContext.java
```

The `Queued()` action is the service invoked by the Task Queued event.

When the `wait()` call returns, the code checks the result of `waitObject.getResult()` which returns information from new task being queued. In the sample, this is the taskURL of the task being started.

Completing a task and waiting for the next task in the workflow

The basic behavior here is similar to that of starting a new process. However, instead of starting a process, the code completes the task and then waits for the next task to be queued. See the code sample in:

```
FormFlowProcessTasks\src\com\webmethods\caf\taskstep1view\
  TaskStep1ViewDefaultviewView.java
```

The `completeTask()` action method addresses the completion requirement.

Ending the workflow from the process

Eventually a workflow task does not transition to a another workflow task, but instead completes the business process. In this case, the original CAF application may be waiting to be notified, but there is no new task to publish a notification. In this case it is the process model that knows the task workflow is complete.

The pub.task.taskclient:notifyFormFlow service in the WmTaskClient package accommodates this use case, enabling you to pass the correlation ID and a result:

```
pub.task.taskclient:notifyFormFlow (correlationID, result)
```

This service serves the same purpose as the `notify()` method of the Java API. It notifies any object waiting for a correlationID and passes a result. The result can be any value that the waiting thread can interpret as a signal to finish the task workflow (for example, it could be NULL, or it could be specific hard-coded value).

For more information about other task services, see the PDF publication *webMethods Task Engine API and Service Reference*.