

## Chapter 6

# Building Workflows

This chapter describes the steps to build a workflow. The functionality of specific work items is described in *Chapter 7, Work Item Library*. The work items and some of the supporting classes are fully compliant JavaBeans components. You can use any compatible tool to assemble your workflows. The examples in this chapter use IBM Visual Age for Java 3.5 to give you an idea of the process. The details will differ depending on the design tool that you use.

This chapter contains the following sections:

Before You Begin on page 77

Creating a Simple Workflow on page 79

Implementing the Workflow on page 83

### Before You Begin

---

Before you assemble a workflow, you must prepare your files and projects:

1. Start Visual Age for Java (VAJ) and go to the Workbench.
2. Import the SDX workflow code into the tool's workspace.

In a default installation, the SDX workflow code is located in `/opt/UMC/wkf/lib` on the workflow host. Import all JAR files contained in that directory.

3. Create four different projects, one for each group of related classes that you import:

One for classes related to Java Mail (*mail.jar*, *pop3.jar* and *activation.jar*)

One for the Netscape LDAP SDK (*ldapjdk.jar*), which contains the classes to access the SDX directory via LDAP

One for the XML-related libraries (*xalan.jar*, *xalanj1compat.jar*, *xerces.jar*, *dsml.jar*)

One for the workflow-specific code (*workflow.jar* and *wf\_lib.jar*)

4. Import all the classes, resources, and source files provided with *wf\_samples.jar* into a project separate from the SDX workflow code.

5. Version each project.
6. Create another project to hold your new workflows.
7. In that project, create a package. Do not use the default package for workflows.
8. Open the Visual Composition Editor (VCE).
9. Add all of the beans described in Table 32 to the palette in the group of your preference. The VCE palette subsequently displays an icon for each bean, as shown in Figure 8.

**Table 32: Work item beans**

Work Item Name	Work Item Type	Description
AND	Synchronization	Synchronizes all its predecessors. If all the input tokens have a proceed result, it will have a proceed output; otherwise, it will have a not proceed output.
Directory Lookup	Regular	Retrieves an object from a directory given its DN and puts its attributes in the output token.
Directory Modify	Regular	Modifies the directory using entries from an XML document, specified by a configurable token parameter.
Directory Query	Regular	Performs an LDAP search in the directory and returns the result as an XML document that is output to a configurable token parameter.
Directory Update	Regular	Updates an object in a directory given its DN, using the attribute values from the input token.
End State	End State	Final state of every workflow.
External Program	Regular	Creates an external program (process) and waits for its completion.
Filter/Pass	Token processor	Filters or lets pass a set of properties from the token.
HTTP	Regular	Performs an HTTP or HTTPS request. The specified URL can be static or dynamically composed by token parameters. The returned data (page) is put in the output token as a parameter.
Lazy Logger	Regular	Sample work item that performs the same task as the Logger work item but takes longer, to demonstrate the functioning of the persistence subsystem and other time-dependent features.
Logger	Regular	Sample work item that prints to the standard output the values of the Name and Address properties of the input token.
MIME Form Encoder	Token processor	Encodes a form in a token parameter using the MIME formats multipart/form-data or application/x-url-encoded.
OR	Synchronization	Synchronizes all its predecessors. If at least one input token has a proceed result, it will have a proceed output; otherwise, it will have a not proceed output.
Receive E-mail	Regular	Receives an e-mail through the e-mail adapter. The adapter should point to a valid mailbox in a POP3 or IMAP server.
Script	Regular	Embeds scripts written in Python. Scripts can be both directly entered or referenced through a filename.
Send E-mail	Regular	Sends an e-mail through an SMTP server.
Start State	Start State	Initial state of every workflow.
Status Logger	Regular	Logs a free-form message in the workflowStatus attribute of a transactional object. The message is appended to the current content together with a new line character that serves as a separator.

Table 32: Work item beans (continued)

Work Item Name	Work Item Type	Description
Token Logger	Token processor	Dumps the token to the console or a file. Useful for debugging.
Token Value Assigner	Token processor	Enables you to assign values to token parameters, whose names and values are specified in the indexed property Assignments. Values can be taken from a token parameter, statically configured, or both.
Token Value Checker	Token processor	Checks whether the specified token parameters have the given values.
XML Decoder	Regular	Interprets an XML document and put the information in the output token.
XML Encoder	Regular	Creates an XML document and puts it in the output token. The document is generated by substituting parameters from the incoming token in special tags in a template XML document.

Figure 8: Sample work item palette (from the VCE after adding the work items to the Workflow group)



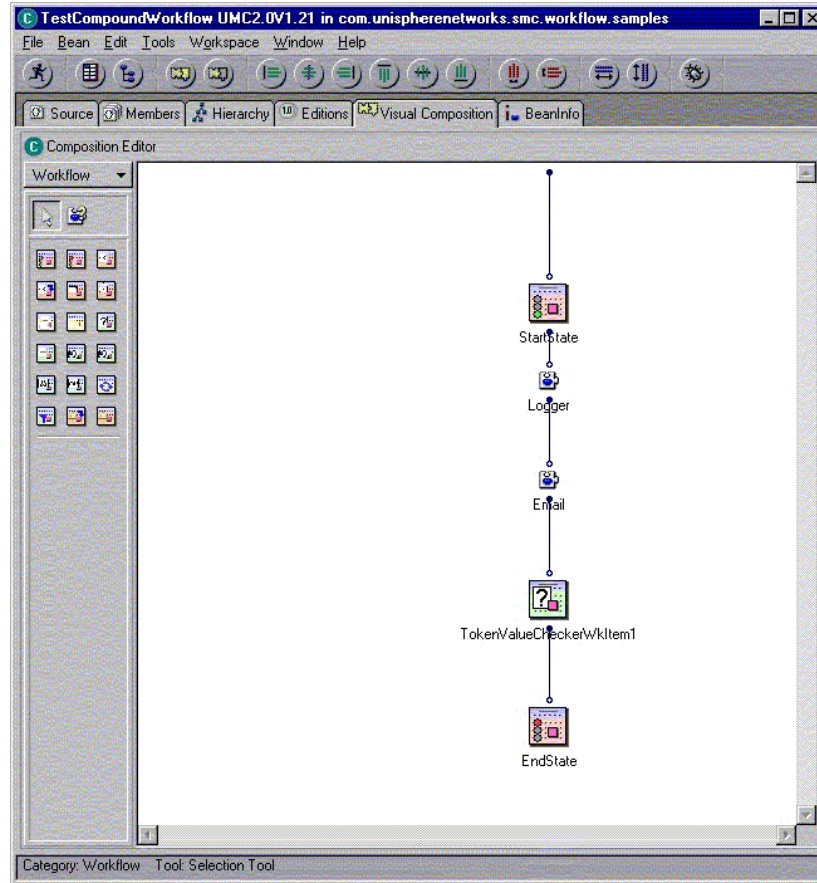
## Creating a Simple Workflow

A very simple workflow, `Test2Workflow`, is provided as an example in the `net.juniper.sgmt.workflow.samples` package. This workflow is composed of only three work items: `StartState`, `LazyLoggerWkItem`, and `EndState`. The `LazyLoggerWkItem` item is in the `wkitemexamples` package. Both packages are in the `wf_samples.jar` file, which is assigned by default to one of the JAR archivist fields. A JAR file can contain various workflows and their respective parameter lists.

The `LazyLoggerWkItem` prints to the standard output the values of two properties obtained from the `User` object related to this service provisioning: `Name` and `Address`. Those parameters are hard-coded inside the work item. The value of the `Name` property will be included in the message the work item logs to the console. You can assign any value to `Name`. `Name` and `Address` are only workflow terms; that is, they are not real attributes of the `User` object. According to the translation table, the attribute names are `cn` for the `Name` and `postalAddress` for `Address`.

Figure 9 shows the sample workflow. The OK output of the StartState work item connects to the LazyLoggerWkItem input. The OK output of LazyLoggerWkItem connects to the EndState work item.

**Figure 9: The Test2Workflow**



### ***Building the Workflow***

Perform the following steps to build this workflow:

1. Select the samples package and open the dialog box to create a new class.
2. Change the base class from `java.lang.Object` to `net.juniper.smgmt.workflow.Workflow`.
3. Check the box `compose class visually`.
4. Click `Finish`.

The tool creates the class and opens the VCE window for that class.

5. Place a `StartState`, an `EndState`, and a `LazyLoggerWkItem` icon in the design area.

6. Connect the StartState and LazyLoggerWkItem icons.
  - a. Right-click on the StartState icon and select Connect | Connectable Features....  
 A dialog box displays the properties that you can use to make a connection.
  - b. Select the property OK, and click Ok.
  - c. Select the LazyLoggerWkItem icon to display a pop-up menu with the options available for that connection.
  - d. Choose *this* from the pop-up menu.  
 A solid green line connects the OK output of the StartState icon to the input of the LazyLoggerWkItem icon.
7. Similarly, connect the LazyLoggerWkItem and EndState icons.
8. Specify the beginning of the workflow.
  - a. Right-click any part of the design area and then click Connect....  
 A dialog box displays the properties that you can use to make a connection.
  - b. Choose startState.
  - c. Select the StartState icon to display a pop-up menu with the options available for that connection.
  - d. Choose *this* from the pop-up menu.
9. Edit the bean properties.
  - a. Double-click on the LazyLoggerWkItem icon to open the bean property sheet, and specify a name by editing the name property.
10. Save the bean.

### **Creating a Parameter List**

After creating a workflow, you must create a parameter list for it. A parameter list specifies the data the system needs from the directory to execute a workflow and the data that will be written back to the directory.

To create a parameter list:

1. In the same package as the workflow, create a new class extending `net.juniper.smg.workflow.WFParameterList` with the name `Test2WorkflowParameterList`. Do not compose the class visually.
2. Edit the constructor of this class, and add the two strings `Name` and `Address` to the `inputParameters` property.

Because the sample workflow does not write anything to the directory, you do not need to specify any output parameters.

3. Save the class.

To make the workflow and parameter list available to the workflow engine, export the `TestWorkflow` and `TestWorkflowParameterList` classes to a JAR file named *test.jar*, in the desired directory. See the VAI documentation to learn how to perform this task.

To configure the workflow engine to see the JAR file:

1. Launch the local configuration tool for the workflow engine.  
`/opt/UMC/wkf/etc/config`
2. Select the Library tab.
3. Enter `test.jar` in one of the JAR Archivist fields.

### ***Deploying the Workflow via the Directory***

Alternatively, you can deploy the workflow via the SDX directory instead of as a JAR file. This method eliminates the need to create the `Test2WorkflowParameterList` class. Instead, you use SDX Admin to create a workflow object named `Test2Workflow` in the Workflows subtree. This object contains the parameter list in string format.

You must specify the qualified class name (that is, including the package name) of the workflow you just built. Finally, import into the directory the class bytecodes, which are obtained from the compiled workflow (.class) file.

## Implementing the Workflow

---

You must restart the workflow engine for the workflow to be available:

```
cd /opt/UMC/wkf/etc  
./wkf stop  
./wkf start
```

Run Workflow Manager to verify that the new workflow is available to the workflow engine.

```
cd /opt/UMC/wkf/bin  
./wkf_manager
```

The name of your workflow should appear in the Workflow Name list.

