Process Builder's Guide

iPlanet™ Process Manager

Version 6.5

816-6354-10
April 2002

Contents

List of Figures

List of Tables

List of Procedures

Preface

About This Manual

This guide, *iPlanet Process Manager*, briefly discusses each component of Process Manager then discusses how to use Process Builder to build process definitions. Read this manual to learn how to build applications for Process Manager.

This manual assumes you have installed Process Builder on your system. For installation instructions, see the *Process Manager Installation Guide*.

It is recommended that you understand JavaScript.

The information in this guide is organized according to the following topics:

Overview of Process Builder Components

The first chapter of this manual provides an overview of the components in Process Builder:

- Chapter 1, "Overview of iPlanet Process Manager"

  See this chapter for information about where to find information about components other than Process Builder.

Using Process Builder

The next nine chapters discuss how to use Process Builder:

- Chapter 2, "Introduction to Process Builder"
- Chapter 3, "Planning an Application"
- Chapter 4, "Creating an Application"
- Chapter 5, "Designing a Process Map"
- Chapter 6, "Defining Groups and Roles"
- Chapter 7, "Defining Data Fields"
- Chapter 8, "Designing Forms"
- Chapter 9, "Using Scripts"
- Chapter 10, "Deploying an Application"

Advanced Scripting and Setting Up Searching

The next three chapters provide more information on writing scripts and setting up searching:

- Chapter 11, "Advanced Techniques for Scripting"
- Chapter 12, "Scripting with EJB Components"
- Chapter 13, "Setting Up Searching"

Sample Applications

The next four chapters describe some of the sample applications included with Process Manager:

- Chapter 14, "The Data Sheet Application"
- Chapter 15, "The Office Setup Application"
- Chapter 16, "The Loan Management and Credit History Applications"
- Chapter 17, "The Insurance Claim Processing Application"

Reference Material

The remaining parts of this guide are reference material:

- Appendix A, "JavaScript API Reference"
- Appendix B, "Reserved Words"
- "Glossary"

Conventions Used in This Manual

File and directory paths are given in Windows format (with backslashes separating directory names). For Unix versions, the directory paths are the same, except slashes are used instead of backslashes to separate directories.

This guide uses URLs of the form:

`http://`*server*`.`*domain*`/`*path*`/`*file*`.html`

In these URLs, *server* is the name of server on which you run your application; *domain* is your Internet domain name; *path* is the directory structure on the server; and *file* is an individual filename. Italic items in URLs are placeholders.

This guide uses the following font conventions:

- The `monospace` font is used for sample code and code listings, API and language elements (such as function names), file names, path names, directory names, and HTML tags.
- *Italic* type is used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
- **Boldface** type is used for glossary terms

Viewing Documentation

For your convenience, Process Manager manuals are provided in both PDF and HTML formats. You can access the documentation from the Help menu of each Process Manager component. You can access context-sensitive documentation by clicking a Help button or link in each Process Manager component.

The location of the documentation in your distribution is at:

`iPM_Install_Dir``/builder/manual`

The documentation is also available at the Sun documentation web site:

`http://docs.sun.com/`

For More Information

Table 1 summarizes the tasks involved in using Process Manager and explains where to go for more information about each task.

Table 1 Summary of Process Manager Components

Do What?	Which Process Manager Component?	Comments
Install Process Manager	Installation component	For more information about installing Process Manager, see the *Process Manager Installation Guide*
Build a process application	Process Builder	Process Builder is a graphical user interface for building process applications. For more information, read the *Process Builder's Guide*.
Perform the steps in a process	Process Express	Process Express is a web-based interface used by the people who perform tasks in a process. It includes a customized worklist for each qualified user as well as a web-based form for each task. For more information about Process Express, see the *Process Express User's Guide*.
Administer a process application	Process Administrator and Business Manager	Process Administrator and Business Manager are a set of web-based interfaces for performing administration tasks such as: creating, deleting, and joining Process Manager clusters, and administering process applications deployed to clusters. For more information, read the *Process Administrator's and Business Manager's Guide*.
Build Java custom data fields and activities in Java. Also use the Java API to programmatically interact with Process Manager.	Process Manager Java classes and API which are available in a JAR file.	Java programmers can build custom data fields and custom activities in Java that can be imported into the Process Builder. Programmers can also create Java applications that embed Process Manager functionality or present customized front ends to Process Manager. Additionally, programmers can create SOAP clients that access web services APIs exposed by Process Manager applications. For more information, see For more information, see the *Process Manager Programmer's Guide*.

NOTE Process Manager runs on top of iPlanet Application Server (iAS). For more information about iAS and other iPlanet products, see the Sun documentation web site at http://docs.sun.com/.

Chapter 1

Overview of iPlanet Process Manager

Process Manager is a system for automating the flow of control for business processes. This chapter briefly discusses process workflow, and gives an overview of how Process Manager can be used to automate the flow of control in business processes.

This chapter has the following sections:

Workflow in Business Processes

Most business processes can be broken down into a series of discrete tasks to be performed by different people or groups of people or to be executed automatically. When one task is completed, the flow of control moves to the next task.

Consider the process for preparing an office for a new employee. Several different activities make up the process -- assigning an office, ordering a computer, installing the phone, installing the computer, and checking that the office furniture has been arranged properly. Some of these tasks need to be performed sequentially, for example, you must order the computer before installing it. Other tasks can be carried out in parallel, for example you don't need to wait for the computer to be ordered before installing the phone. Different people perform different tasks -- the purchasing department orders the computer, but the Information Systems department installs it when it arrives.

For another example, consider a process for submitting an insurance claim. The players and activities in this case include the customer submitting the claim; the computer retrieving details of the claimant's policy from a database; the insurance underwriter approving or denying the claim; and the finance department mailing the check.

Process Manager allows you to create web-based applications that define the different tasks in a process, specify who should perform them, and map out how the process flows from one task to another. The following list summarizes the tasks involved in using Process Manager:

- "Designing a Process"

  Using the Process Builder component, process designers create and deploy an application that maps the steps in a business process.

- "Performing Tasks in a Process"

  The Process Express component provides a web-based interface that aids the people assigned to perform a task.

- "Administering Processes"

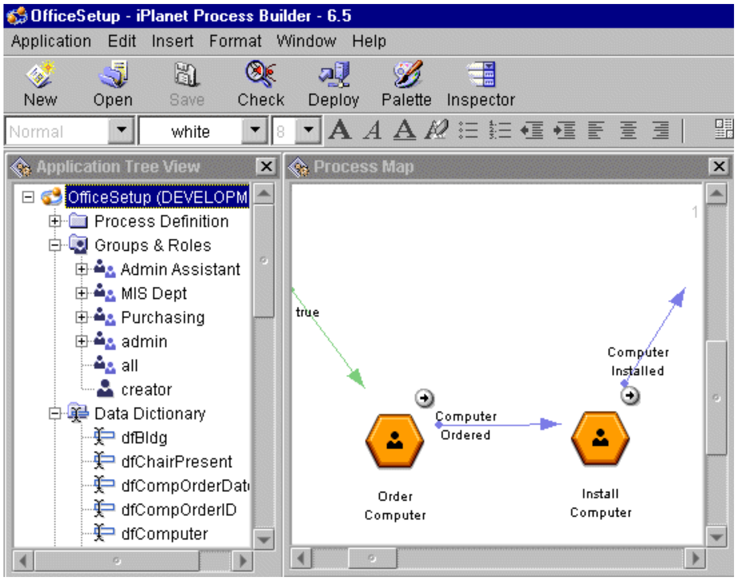
  The Process Administrator component provides a web-based interface for administrators to administer process applications, such as shutting down processes.

Designing a Process

Using Process Builder, a graphical user interface with drag and drop capability, you build a process application that controls the flow of a process. The process can contain tasks, or work items, that require human intervention such as approving a purchase request, as well as activities that can be completely automated such as retrieving data from databases and writing log files.

Figure 1-1 illustrates Process Builder.

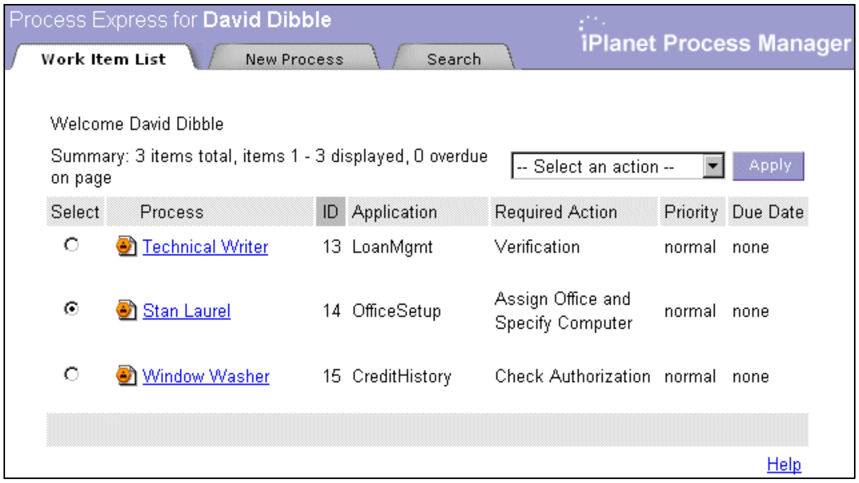
Figure 1-1 Process Builder



Performing Tasks in a Process

When the process is deployed, Process Manager routes tasks at each step in the process to the users that need to perform them. Process Manager uses a web-based interface, called the Process Express, to interact with the users involved in the process. The Process Express displays a web-based work list that informs users of tasks that have been assigned, as illustrated in Figure 1-2.

Figure 1-2 The Worklist in the Process Express



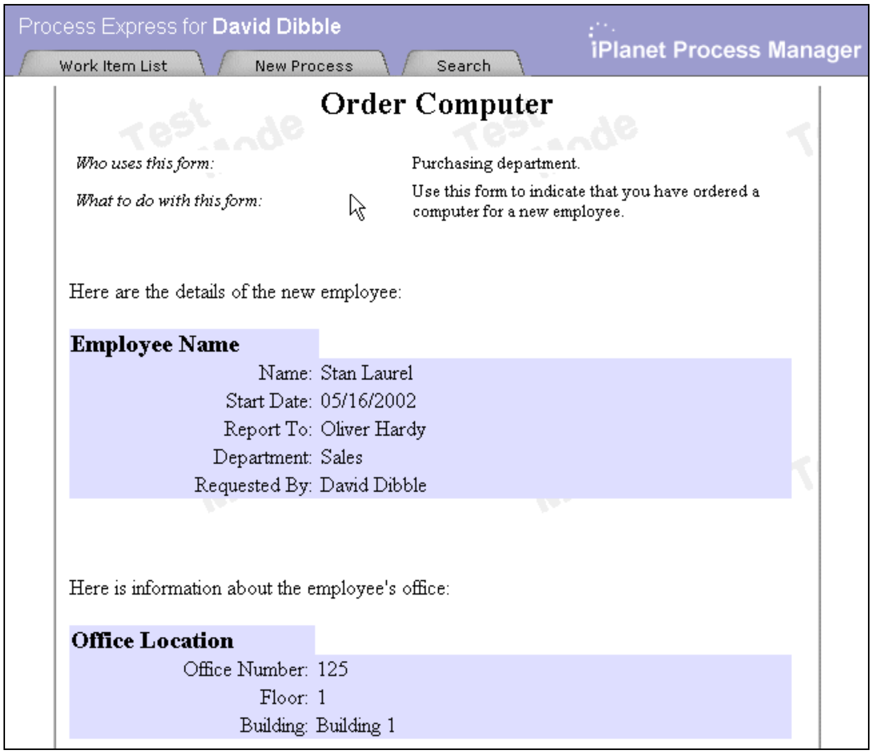
Welcome David Dibble

Summary: 3 items total, items 1 - 3 displayed, 0 overdue on page

Select	Process	ID	Application	Required Action	Priority	Due Date
○	Technical Writer	13	LoanMgmt	Verification	normal	none
●	Stan Laurel	14	OfficeSetup	Assign Office and Specify Computer	normal	none
○	Window Washer	15	CreditHistory	Check Authorization	normal	none

When an assignee is ready to perform the task, they click on the task name. Process Express displays a form that provides the data needed to perform the task, as illustrated in Figure 1-3.

Figure 1-3 Example of a work item form



Order Computer

Who uses this form:	Purchasing department.
What to do with this form:	Use this form to indicate that you have ordered a computer for a new employee.

Here are the details of the new employee:

Employee Name

Name:	Stan Laurel
Start Date:	05/16/2002
Report To:	Oliver Hardy
Department:	Sales
Requested By:	David Dibble

Here is information about the employee's office:

Office Location

Office Number:	125
Floor:	1
Building:	Building 1

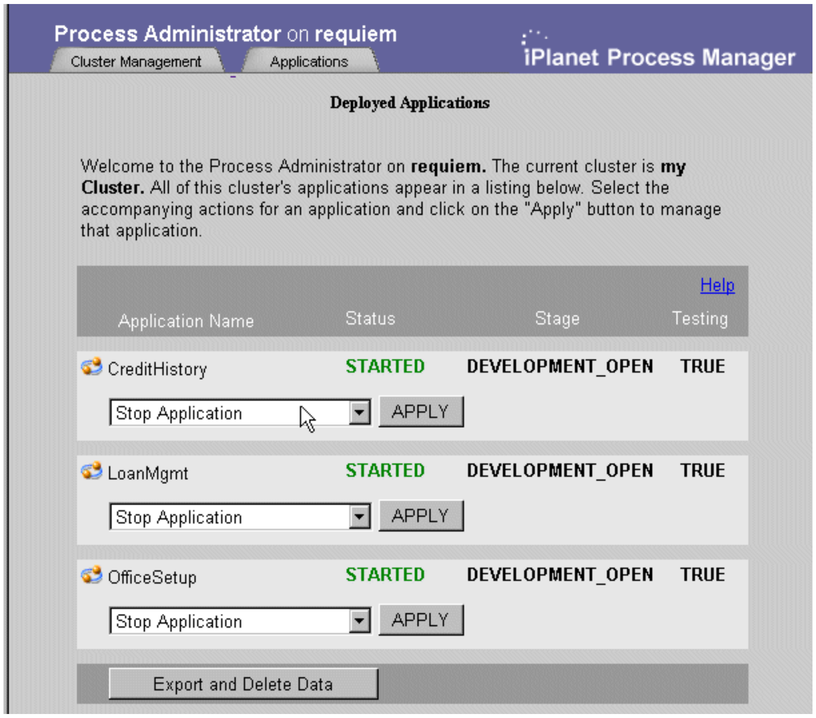
When an assignee has performed the task, they enter any data needed to complete the task and then submit the form. The work item automatically disappears from the assignee's work list. The process moves on to the next step and the next task shows up in the worklist of the appropriate user.

Process Manager automatically performs any automated tasks that do not require human intervention.

Administering Processes

Process Manager includes another set of web-based interfaces, called Process Administrator and Business Administrator, for performing administrative duties, such as shutting down deployed processes. Figure 1-4 shows one of the administrative interfaces.

Figure 1-4 Process Administrator



Deployed Applications

Welcome to the Process Administrator on **requiem.** The current cluster is **my Cluster.** All of this cluster's applications appear in a listing below. Select the accompanying actions for an application and click on the "Apply" button to manage that application.

Application Name	Status	Stage	Testing
CreditHistory	STARTED	DEVELOPMENT_OPEN	TRUE
LoanMgmt	STARTED	DEVELOPMENT_OPEN	TRUE
OfficeSetup	STARTED	DEVELOPMENT_OPEN	TRUE

Summary of Process Manager Components

Table 1-1 summarizes the tasks involved in using Process Manager and describes where to go for more information about each one.

Table 1-1 Summary of Process Manager Components

Do What?	**Which Process Manager Component?**	**Comments**
Build a process application	Process Builder	Process Builder is a graphical user interface for building process applications. For more information, read the *Process Builder's Guide*.
Perform the steps in a process	Process Express	Process Express is a web-based interface used by the people who perform tasks in a process. It includes a customized worklist for each qualified user as well as a web-based form for each task. For more information about Process Express, see the *Process Express User's Guide*.
Administer a process application	Process Administrator and Business Administrator	Process Administrator and Business Manager are a set of web-based interfaces for performing administration tasks such as: creating, deleting, and joining Process Manager clusters, and administering process applications deployed to clusters. For more information, read the *Process Administrator's and Business Manager's Guide.*
Build Java custom data fields and activities in Java. Also use the Java API to programmatically interact with Process Manager.	Process Manager Java classes and API which are available in a JAR file.	Java programmers can build custom data fields and custom activities in Java that can be imported into the Process Builder. Programmers can also create Java applications that embed Process Manager functionality or present customized front ends to Process Manager. Additionally, programmers can create SOAP clients that access web services APIs exposed by Process Manager applications. For more information, see For more information, see the *Process Manager Programmer's Guide.*

Providing the backend support to these interfaces is the Process Manager Engine, which performs the "behind the scenes" work of managing the process applications as well as performing and routing tasks in individual processes.

Process Manager requires various other software components to be installed, such as an application server and a directory server. See the *Process Administrator's and Business Manager's Guide* for more details about these additional components.

This manual discusses how to use Process Builder to design and deploy applications, and describes in detail the sample applications that ship with Process Builder.

Chapter 2

Introduction to Process Builder

This chapter provides an overview of Process Builder, along with basic instructions for starting and using it.

This chapter contains the following sections:

- "About Processes and Process Builder" on page 41
- "About Applications" on page 45
- "Configuring Builder for Secure Deployment" on page 47
- "Configuring Builder for Deployment Through a Proxy" on page 50
- "Configuring Builder for Internationalization Support" on page 50
- "Starting Process Builder" on page 54
- "Using Process Builder" on page 56

About Processes and Process Builder

This section contains the following topics:

- "What is a Process?"
- "What is Process Builder?"
- "Who Should Use Process Builder?"
- "What is a Process Instance?"

What is a Process?

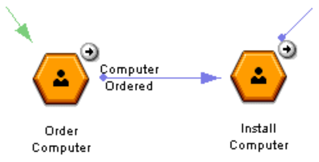
A process is a series of steps that can be completed by one or more end users who participate in the process. For example, your company may have a process for requesting vacation time. The process participants would be an employee who makes the request, the manager who must approve it, and the payroll department that updates the employee's records.

What is Process Builder?

Process Builder is a graphical tool used to create, maintain, and deploy web-based process applications. These applications can handle forms-based processes and can also integrate with external systems.

You use Process Builder to design and create a process map. This map defines the tasks involved in the process and specifies the flow of control through the process from one task to another. The figure below shows an example of two activities in a map:

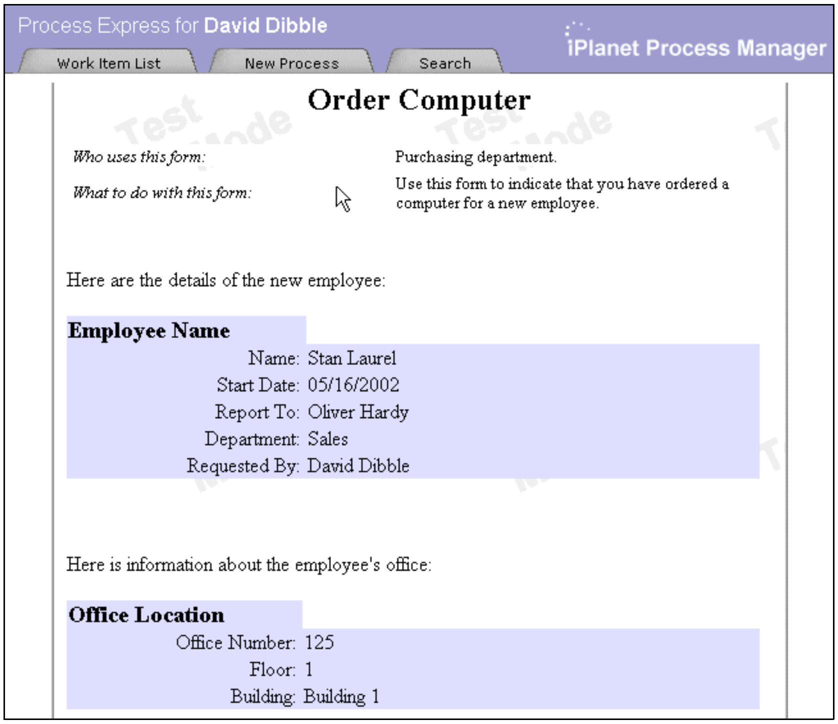
Figure 2-1 Two activities in a process map



Process Builder allows you to define various kinds of activities in a process, including: activities that require human effort, activities that are performed automatically, activities that can run in parallel, and subprocesses, which are tasks that are complete multi-step processes in themselves.

Process Builder is also where you create the forms associated with the various tasks. Figure 2-2 shows the form that is associated with the "Order Computer" task:

Figure 2-2 Order Computer Form



Order Computer

Who uses this form: Purchasing department.

What to do with this form: Use this form to indicate that you have ordered a computer for a new employee.

Here are the details of the new employee:

Employee Name

Name:	Stan Laurel
Start Date:	05/16/2002
Report To:	Oliver Hardy
Department:	Sales
Requested By:	David Dibble

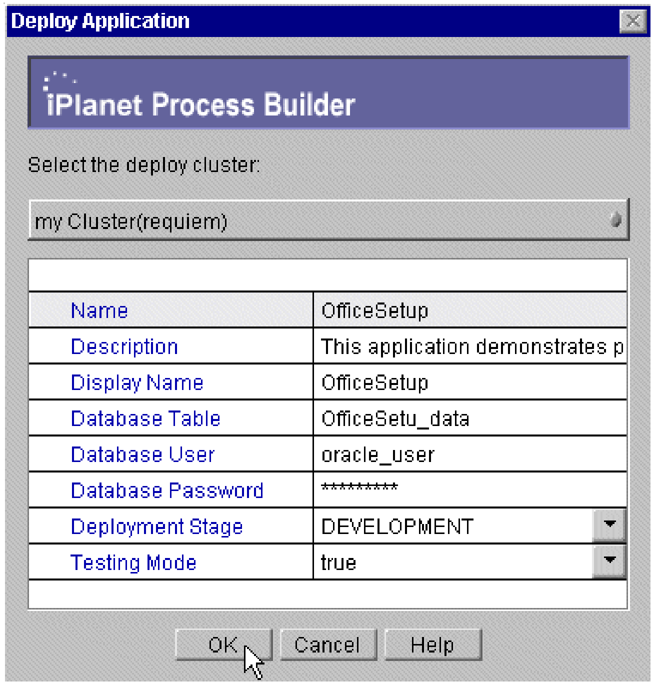
Here is information about the employee's office:

Office Location

Office Number:	125
Floor:	1
Building:	Building 1

When you have defined a process in the Builder, you deploy it to a server where Process Manager is running. Deployment is very simple, all you need to do is to enter a few details in a dialog box, as shown in Figure 2-3.

Figure 2-3 Deployment Dialog Box



Deploy Application

iPlanet Process Builder

Select the deploy cluster:

my Cluster(requiem)

Name	OfficeSetup
Description	This application demonstrates p
Display Name	OfficeSetup
Database Table	OfficeSetu_data
Database User	oracle_user
Database Password	*********
Deployment Stage	DEVELOPMENT
Testing Mode	true

Who Should Use Process Builder?

Process Builder is for people who design and deploy Process Manager applications. We refer to these people as process designers. As a process designer, you need to know JavaScript. In addition, if you want to extend Process Manager applications so that they can access external systems, you need to program in Java.

What is a Process Instance?

You create a separate application for each process. For example, submitting a vacation time-off request and submitting an expense report are two different applications.

When you deploy applications to a cluster, end users can access them. For more information about clusters, see the *Process Administrator's and Business Manager's Guide*.

When users submit their first request in a Process Manager application, they initiate what is called a process instance. Each user's request generates a unique process instance. The point where a user initiates a process instance is called an entry point. The process instance ends at an exit point.

About Applications

This section contains the following topics:

- "Creating an Application"
- "Deploying an Application"
- "Sample Applications"

Creating an Application

To create a process application using Process Builder, you create a graphical view of the steps in the process, from beginning to end. This view is called the process map. You must also define everything that relates to the steps, such as their forms, data fields, scripts, and access to forms.

In summary, you must create the following application elements:

- the process map, which defines the steps in the process
- the data dictionary, which captures process data
- the HTML forms that are displayed at each step
- the scripts that perform functions within an application
- the groups and roles of the people who will be using the application
- the content store, which specifies where files associated with the process are stored

Application elements are represented in a window called the application tree view. This window resembles a typical file system view: it uses folders to represent element categories, and the folders contain the corresponding application elements. Initially, the folders in the application tree view are empty, or they contain default values.

To create the steps in the process, you drag icons from the Palette onto a process map. You connect steps by drawing arrows between them. These connections are called transitions.

As you construct the process map and insert new data fields, forms, and scripts, these items are also added to the application tree view. In this way, the folders are filled with items represented on the process map. Since all application elements are represented in the application tree view, you can easily access and edit any element's properties.

Deploying an Application

Once you have created an application, you can save it to a local folder, or deploy it to a cluster.

When you save the application on your local machine, all your definitions (the process map items, the data dictionary, the forms, etc.) are stored in a local applications folder in the `builder` folder.

Deployed applications are stored in clusters. When you launch Process Builder, you get a list of all the clusters specified in the `preferences.ini` file, as well as a list of any applications saved locally in the default directory.

When you deploy the application, you copy all its definitions to the cluster as entries in the configuration directory. There are two deployment stages: development and production. Deploy to development if you are still testing the application and would like to be able to make the fullest possible range of changes. Deploy to production when you are ready to give end-users access to the application.

For information on deploying applications, see Chapter 10, "Deploying an Application."

Sample Applications

Process Builder includes a set of sample applications you can use as models when you create your own applications. For information on using and configuring the sample applications shipped with Builder, see the following chapters:

- Chapter 14, "The Data Sheet Application"
- Chapter 15, "The Office Setup Application"
- Chapter 16, "The Loan Management and Credit History Applications"
- Chapter 17, "The Insurance Claim Processing Application"

Configuring Builder for Secure Deployment

The following sections describe how to set up your development environment to support deployment of process applications using https:

- "About https Deployment Support for Process Builder"
- "Configuring the Server (iWS and iPM)"
- "Enabling Content Store Over https"

About https Deployment Support for Process Builder

Process Manager is automatically configured to enable secure deployment. There are, however, some configuration issues that you must address to ensure that your developers deploy their applications to an iPM cluster securely. This section describes those configuration issues. Process Manager uses the Java Secure Socket Extension (JSSE) libraries and the cacerts database for secure deployment. By default, the JSSE libraries are installed in the <*iPM_install_directory*>`/builder/lib` directory.

Process Builder and the Command Line Deploy Tool (`pmdeploy`) support the deployment of process applications to iPM clusters over a web server that uses the https protocol. The https protocol uses a digital certificate to encrypt and send documents and files securely.

iPlanet Process Manager supports handshake between the server and the client, where the server presents a digital certificate (server certificate) but does not require a certificate (client certificate) from the client.

During the deployment step, the server presents a certificate to the client (Builder/`pmdeploy`) that the client must accept. This certificate ensures the client deploys to the correct server.

The client consults its certificate store for an acceptable certificate match. If the certificate presented is not in the store, the client does not allow the communication to go through and the application is not deployed. If the certificate presented is in the store, the communication continues and the application is deployed.

TIP When communication is rejected in the Process Builder you receive a message indicating that you are not able to connect to the web server. For best results when deploying applications, run Builder from a console window using `Builder.bat` or `Builder.sh` and consult the console window for any exceptions.

Configuring the Server (iWS and iPM)

1. Install the server certificate on the web server and turn on security.

   This involves setting up a web server instance to run in https mode. See "Securing Your Web Server" in the *Web Server Administrator's Guide* for instructions on how to do this.

2. Install the iAS Web Connector Plug-in on the web server instance and configure it to point toward the iAS instance on which iPM is running.

   For more information about the Web Connector Plug-in, see "Configuring the Web-Connector Plug-in" in the *iPlanet Application Server Administrator's Guide*.

3. Create an iPM cluster using this (https) web server instance.

   The newly created iPM cluster now uses the https web server instance for future process application deployments.

Enabling Content Store Over https

The Content Store helps iPM users upload and download file attachments. The file attachment Java applet communicates to the server via http and https. (The https protocol works only on Netscape browsers.)

Once a file is uploaded to the web server, the process application may require access to it. In this case, the process application needs to use the https protocol to access the uploaded file if the web server instance is https enabled.

➤ **To configure the Application Server environment to make https handshake possible**

1. Install the JSSE jars as a standard Java extension package.

   To do this, copy `jnet.jar`, `jcert.jar`, `jsse.jar` to `$JAVA_HOME/jre/lib/ext`. The default `JAVA_HOME` is <*ias_install_dir*>`/ias/usr/java`.

   You can locate the JSSE jar files in the <*iPM_install_directory*>`/builder/lib` directory.

2. (NT-specific) Modify Java Args On NT.

   a. Open the iAS Registry using kregedit and locate the `Java_Args` key: `Application Server\6.5\Java\Java_Args`.

   b. Add the following Java Arg to the key: `-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol`

3. (Solaris-specific) Modify Java Args On Solaris.

   a. Open <*ias_install_dir*>`/ias/env/iasenv.ksh`.

   b. Locate the `JAVA_ARGS` variable and add the following Java Arg: `-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol`

4. Update the JSSE provider (SunJSSE) in the Java installation.

   a. Open the file <*java_install_dir*>`/jre/lib/security/java.security`.

   b. Add SunJSSE as the second preferred cryptographic service provider as follows:

```
security.provider.1=sun.security.provider.Sun
# Keep above line intact
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

Configuring Builder for Deployment Through a Proxy

If you want to deploy Process Manager applications through a proxy server, you must enable proxy support for Process Builder. Process Manager supports deployment through normal (http) and secure (https) protocols.

➤ **To configure Builder for deployment through a proxy server**

1. Open the preferences.ini file located in the following directory:

   <*iPM_install_directory*>`/builder`

2. Locate the line that references proxy support `#Proxy Support`
3. Uncomment `useProxy` and set it to true.
4. Uncomment `proxyHost` and set the proxy host you want to use.
5. Uncomment `proxyPort` and set the proxy port you want to use. If you do not specify a proxy port number, Builder uses the default port of 80.

NOTE You must set the `useProxy` to true. If you specify a proxy host and proxy port, but fail to set `useProxy` to true, Builder ignores your host and port settings.

Configuring Builder for Internationalization Support

This section contains the following topics:

- "Enabling Non-ASCII Character Display in Builder"
- "Enabling the InputMethodEditor for Non-ASCII Character Display in Builder"
- "Operating Character Set Changes"
- "Configuring Locale Settings for Applications Using Mulitbyte Characters"

Enabling Non-ASCII Character Display in Builder

To enable non-ASCII character display in Builder, you must modify the `preferences.ini` file located in the `<iPM_install_directory>/builder` directory.

➤ **To modify the preferences.ini file for non-ASCII character display**

1. Open the `preferences.ini` file in a text editor.
2. Uncomment the following entries:

   ```
   #NormalTextFont=sansserif:plain:12
   #SmallTextFont=sansserif:plain:10
   #NormalBoldTextFont=serif:bold:14
   #HeadingTextFont=serif:bold:16
   #FixedFont=monospaced:plain:14
   #TitleTextFont=sansserif:bold:12
   ```

   To uncomment entries, remove the '#' from the beginning of each line.
3. Save the `preferences.ini` file.

Enabling the InputMethodEditor for Non-ASCII Character Display in Builder

The InputMethodEditor allows non-ASCII character input and display in three components of the Process Builder:

- Node name editor on Process Maps
- Name and Description fields in the node property editor
- Description and Display Name fields in the application property editor

If Process Builder is running on an OS with the English locale set, the InputMethodEditor is disabled by default. To enable the InputMethodEditor for non-ASCII character display, uncomment the following line in the `preferences.ini` file:

```
#useInputMethodEditor=true
```

If Process Builder is running on an OS with a non-English locale set, the InputMethodEditor is enabled by default. To disable the InputMethodEditor, uncomment the following line in the `preferences.ini` file:

```
#useInputMethodEditor=false
```

Operating Character Set Changes

Process Builder creates forms using a character set of UTF-8. All forms created by Builder include the following meta tag:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=UTF-8">
```

NOTE If you decide to open an html form created in Builder in another editor, make sure the editor you use does not change the character set in the meta tag.

Before you deploy an application, make sure the character encoding of the forms (html files) matches the character set of the OS where the web server is running.

If you develop an application on NT using the Japanese character set and want to deploy it to Solaris Japanese operating in a ja locale, you must convert all html files in the application from Shift_JIS (OS character set of NT Japanese) to EUC-JP (OS character set of Solaris Japanese for ja locale).

Some tool applications support character set conversion of files. The JDK provides the command line tool `native2ascii`.

➤ **To convert a file encoded in Shift_JIS to EUC-JP using native2ascii**

1. Call the command

   ```
   native2ascii shiftJIS.html unicode.html
   ```

2. Call the command

   ```
   native2ascii -reverse -encoding EUC-JP unicode.html eucJP.html
   ```

   `shiftJIS.html` encoded in `Shift_JIS` becomes `eucJP.html` encoded in `EUC-JP`.

To run the iPM application using the new character set, the following items must be set for the iPlanet Application Server, Database Server, and Database Client.

- Set the International key to 1 in the Application Server Registry:
    - **a.** Start kregedit by typing "kregedit" at the command line.
    - **b.** Open the `SOFTWARE\iPlanet\Application Server\6.5\NLS\INTERNATIONAL` key.
    - **c.** Change the value to 1, then click OK.
    - **d.** Restart the Application Server.
- Set the operating character set for the DataBase Server to UTF-8
- Set the operating character set for the DataBase Client to UTF-8

Configuring Locale Settings for Applications Using Mulitbyte Characters

When forms (html files) contain multibyte characters, the locale for both the development system where Builder is used to develop an application and the target system where the iPlanet Application Server is running must have the same locale setting. The character set used in forms must also match the character set of the locale of the target server system.

The following combinations are valid for a Japanese application:

- Develop on NT Ja, deploy to iAS running on NT Ja

  No extra work is involved.
- Develop on NT Ja, deploy to iAS running on Solaris Ja with a ja locale using EUC-JP as the OS charset

  Before deployment, you must convert all forms to EUC-JP. You must declare the UTF-8 character set in the meta tag.
- Develop on Solaris Ja with a ja locale, deploy to Solaris Ja with ja locale

  No extra work is involved.
- Develop on Solaris Ja, deploy to iAS running on NT Ja

  Before deployment, you must convert all forms to Shift_JIS. You must declare the UTF-8 character set in the meta tag.

Starting Process Builder

Before you can use Process Builder, you must install it on your local system. For installation instructions, see the *Process Manager Installation Guide*.

This section contains the following topics:

- "Editing the preferences.ini File"
- "Starting Process Builder on Windows NT"
- "Starting Process Builder on Solaris"

Editing the preferences.ini File

The `preferences.ini` file specifies the available clusters and the corporate user directory, as well as some of the folders that appear by default in your initial Process Builder window. The installer puts `preferences.ini` in the `builder` folder.

After the administrator creates a cluster, you need to add lines to the file specifying the cluster(s) and the corporate user directory you are accessing. The Process Administrator interface returns and displays these lines when a cluster is created. Copy them from the successful cluster creation screen and paste them in your `preferences.ini` file. Otherwise, use the examples below to edit your `preferences.ini` file by hand.

Specifying a Cluster

To access a cluster from Process Builder, add a line to the `preferences.ini` file in the following format. (Although shown below as multiple lines, be sure to type it as one line in the file, with no space after the @ symbol.)

```
cluster = ldap://Unrestricted_User:Password@
Configuration_Directory:port/clusterDN
```

For example:

```
cluster = ldap://cn=Directory Manager:password@
directory.mozilla.com:389/cn=cluster, o=airius.com
```

Specifying a Corporate Directory

To access a corporate user directory from Process Builder, add a line to the `preferences.ini` file in this format:

```
corp_dir = ldap://corporate_Directory_Server:port/corp_Directory_Suffix
```

For example:

```
corp_dir = ldap://directory.mozilla.com:389/o=airius.com
```

You may want to add a `corp_dir` line to your `preferences.ini` file even if you do not know what cluster you will be deploying to. By adding a `corp_dir` line, the corporate user directory is available to you as you design applications, even without a deployment cluster.

Typically, your corporate directory does not require user authentication. However, if your corporate directory requires authentication, you can add a line in the following format (on one line):

```
corp_dir = ldap://Unrestricted_User:Password@
corporate_Directory_Server:port/corp_Directory_Suffix
```

CAUTION The `preferences.ini` file also includes parameters that define the Java Virtual Machine. Do not modify this part of the file.

Adding or Changing Local Application Folders

You can also change some of the default local application folders in `preferences.ini`. The file ships with the following lines defining folders:

```
local_folder = Sample Applications; Samples
```

```
local_folder = Scratch Folder; Tmp
```

These two lines define two folders in your builder folder. The Samples folder stores the sample applications that ship with the product, and the scratch folder is a folder you can use for temporary content.

To add additional folders, use the following syntax:

```
local_folder = name_of_folder; pathname_of_folder
```

The pathname assumes you start in the builder folder.

Starting Process Builder on Windows NT

When you install Process Builder on your Windows NT machine, you choose a folder to install it in. By default, this folder is *IAS_Root*`/builder, IAS_Root` indicates the root directory of your iPlanet Application Server installation.

To start Process Builder, go to the folder where you installed Process Builder and double-click on the `PMBuilder.exe` file. This file launches Process Builder.

Starting Process Builder on Solaris

When you install Process Builder on your Solaris machine, you choose a folder to install it in. By default, this folder is `IAS_Root/builder`, where `IAS_Root` indicates the root directory of your iPlanet Application Server installation.

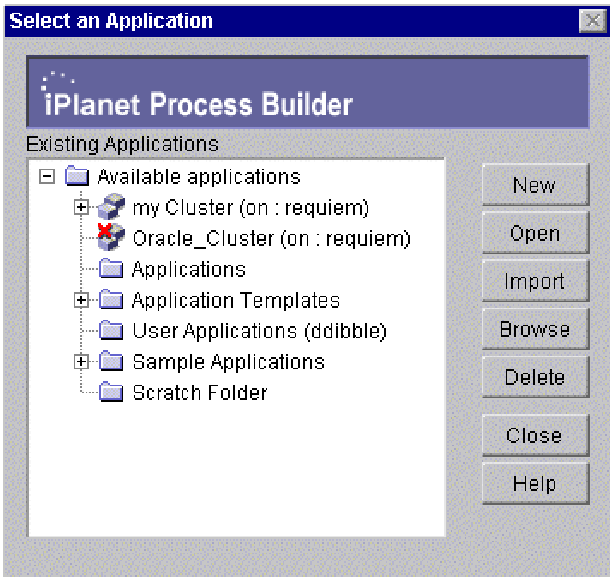
To start Process Builder, go to the directory where you installed it and type the following at the command prompt:

```
PMBuilder.sh
```

Using Process Builder

When you start Process Builder, you see a window like the one shown in Figure 2-4.

Figure 2-4 Select an Application window



In the left pane, this window lists all clusters, the applications deployed to them, and applications saved locally in the application directories. The applications in the clusters have labels indicating whether they were deployed in the development stage or in the production stage.

Before a cluster appears in this window, the Process Manager administrator must create it, and you must have edited your `preferences.ini` file to contain the correct cluster information. For more information on clusters in your environment, contact your administrator. Some of the local folders are also defined in the `preferences.ini` file.

For more information about the `preferences.ini` file, see "Editing the preferences.ini File" on page 54.

The following local folders are provided by default:

Applications The main folder for storing applications.

Application Templates The folder where you store applications that you base other applications on. For example, if your applications share many elements in common, you might want to designate a template application, save it to this folder, and use it as a basis for future applications. A sample template application, Default, is included in this folder as an example.

User Applications A folder for applications stored by user. On Unix, this folder is your default user home directory.

Sample Applications A folder that contains all the applications shipped as samples with Process Builder. This folder is defined in your `preferences.ini` file.

Scratch Folder A folder for storing temporary, or scratch, versions of your applications. This folder is defined in your `preferences.ini` file.

The right pane contains the following buttons:

New Creates a new application.

Open Opens the application selected in the tree.

Import Use this button to import an existing application into your Process Builder. The application must be a .zip file, for example, an application zipped and sent via email. The import function unzips the application into the selected folder and opens the application for editing.

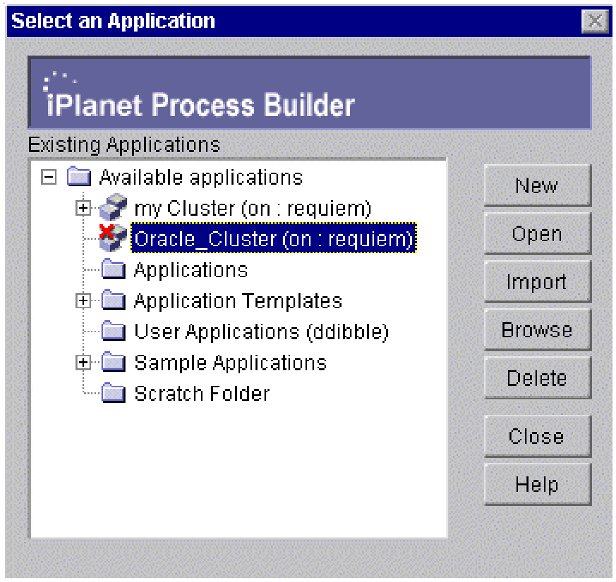
Browse Navigates through the file system to open applications stored outside your local applications folders. Choose the LDIF file of the application you want to open. (LDIF is a file format for storing directory entries). LDIF files have the same names as the application and the folder the application is stored in.

Delete Deletes the selected application. You cannot delete applications that have been deployed. Deployed applications can only be deleted by the administrator.

Cancel Closes the window, but leaves Process Builder running.

Sometimes a cluster icon has a red "x" in it, as shown in Figure 2-5:

Figure 2-5 Cluster error



This icon indicates that Process Builder is unable to access a cluster, and appears when Process Builder cannot access the configuration directory. If you see this icon:

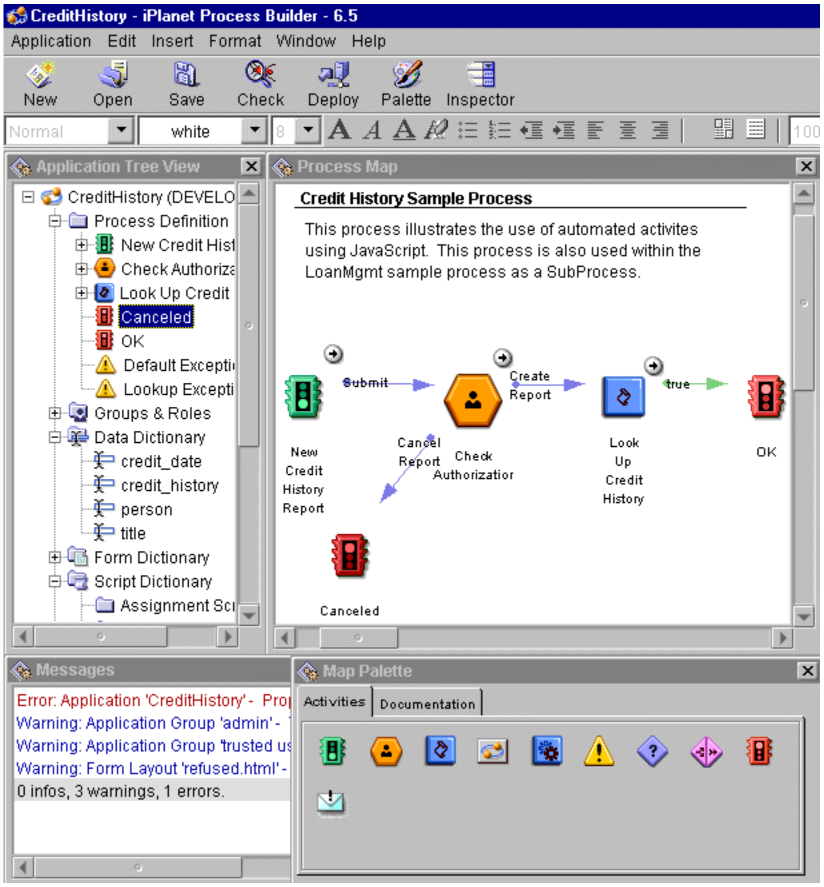
- verify that the cluster information in `preferences.ini` is correct.
- verify that the configuration directory is up and running

Your administrator can also help you solve cluster problems.

Process Builder Applications

Figure 2-6 shows a sample view of Process Builder when you open an existing application (the CreditHistory application in this case).

Figure 2-6 A Process Builder application



This process illustrates the use of automated activites using JavaScript. This process is also used within the LoanMgmt sample process as a SubProcess.

```
Error: Application 'CreditHistory' - Pro
Warning: Application Group 'admin' -
Warning: Application Group 'trusted us
Warning: Form Layout 'refused.html' -
0 infos, 3 warnings, 1 errors.
```

Within each application, Process Builder provides the following views and tools:

- Application Tree View
- Process Map
- Map Palette
- Messages window
- main toolbar
- menu commands
- HTML editor toolbar

You use these views and tools to create and edit applications. In addition, some commands are available by right-clicking on an element in the application tree view or process map.

Application Tree View

The application tree view lists all the elements in an application. Each application has a set of default elements, shown as a set of standard folders and icons. When you create a new application, the element folders are empty or contain defaults. As you construct the process map and insert new data fields, forms, and scripts, the folders fill up with subitems that you can directly access at any time to inspect or customize their properties.

These are the default folders and icons:

Process Definition This folder contains all the items in an application. See "Map Palette" on page 61 for more information on these items.

Groups and Roles This folder contains the application's groups and user roles. See Chapter 6, "Defining Groups and Roles" for more information.

Data Dictionary This folder contains the data fields created for the application. These fields define data that is to be captured when users run the application. See Chapter 7, "Defining Data Fields" for more information.

Form Dictionary This folder contains the HTML forms created for the application. Forms define how information is presented to a user. See Chapter 8, "Designing Forms" for more information.

Script Dictionary This folder contains the JavaScript scripts available to the application, organized by script type. Scripts are either predefined or created by you. See Chapter 9, "Using Scripts" for more information.

Form Access This icon brings up a window which lets you specify which HTML form is displayed for a specific role during each step. You set up access to forms by dragging forms from the application tree view into the window. See Chapter 8, "Designing Forms" for more information.

Content Store The content store icon brings up a window where you specify the URL on the Enterprise Server where you store attached files, and also the public user that can access these files. When end users attach files to forms as they complete steps in the process, the files are stored in the content store. See Chapter 7, "Defining Data Fields" for more information.

Process Map

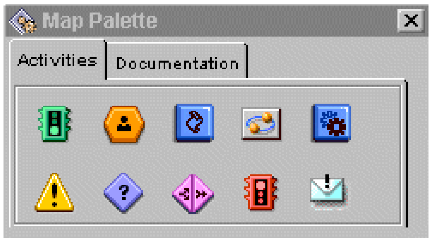
The process map provides a visual representation of an application. It shows the steps needed to complete the process, and how they are connected to each other.

When you create a new application, the Process Map window is blank. If you are revising an existing process, the current version of the process appears in the Process Map window. To design a process in the Process Map, you drag activity icons from the Map Palette and connect them with transitions.

Map Palette

Figure 2-7 shows the Map Palette, also called the palette. When you create a new application or when you open an existing one, the palette is open by default. You can also click the Palette button to open it.

Figure 2-7 The Map Palette



By default, the palette has two tabs: Activities and Documentation. If you use custom activities, you may also have one or more custom tabs on your palette. For more information on adding custom palettes, see "Adding a Custom Palette" on page 114.

Activities Tab

The Activities tab displays icons that you can drag onto the process map to design a process application. Each of the palette items represents a type of step in the process, a notification, or an error response. Table 2-1 shows the icons on the palette and what they represent.

Table 2-1 Icons in the Activities tab

Icon	Description
	Entry Point. A point at which a user can initiate a process. An application can have several entry points. For example, if the first few steps create an ID number for the user, a returning user who already has an ID can skip those steps.
	User Activity. A step within the process that requires a user to perform a task. Each user activity has an assigned user who performs the task (assignee) and a form the user needs to fill out in Process Express. After you place activities in the process map, you define the sequence in which they are to be executed by connecting them with transition lines.
	Automated Activity. An automated step performed through a JavaScript script without user intervention.
	Subprocess. A step that calls a process from within another process. The process that calls the subprocess is considered to be the main, or parent process, and the subprocess is considered to be its subordinate or child process. A parent process can have several children processes, each of which is a stand-alone process complete with entry and exit points.
	Exception Manager. A step that allows the administrator to intervene manually if errors occur when users run the application.
	Decision Point. A conditional step that causes the process to use different steps based on a condition. For example, you might have a decision point that directs the process to different steps depending upon the cost of an item.
	Split-Join (parallel processing). A step within the process that branches in two or more branches so that two or more activities can execute in parallel.
	Exit Point. A step at which the process ends. An application can have several exit points. For example, in a vacation time off request application, an exit point could be the approved vacation request, and another could be a vacation that was not approved.
	Custom Activity. A step at which a Process Manager application connects to external components or services.

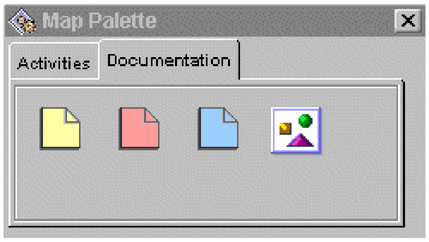
Table 2-1 Icons in the Activities tab

Icon	Description
✉	**Notification.** An email notification that is triggered when a user activity is started. The email can serve many purposes. For example, it can inform the person who started the process or other users of the process's progress.

Documentation Tab

Use this tab (shown in Figure 2-8) to drag documentation notes or graphics to the process map.

Figure 2-8 Documentation Tab on the Map Palette



For example, different developers may include notes on a process map coded by color, or the notes could explain features of the process for use when you roll-out the process.

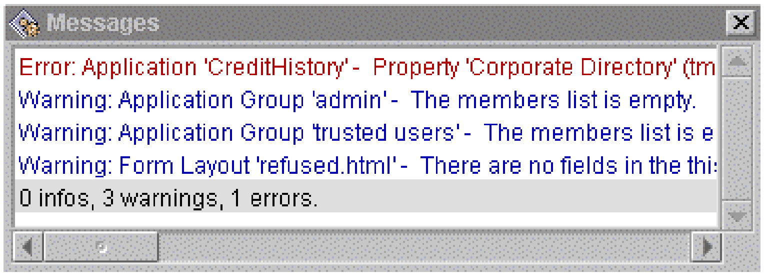
You can also add images to a process map by dragging the image icon to the map and then browsing to the image you want to include.

Messages Window

The Messages window is where error, warning, and information messages are displayed when you check the application syntax using the Check button. The syntax is also checked automatically when you attempt to deploy an application.

The window is blank until the syntax is checked, after that it contains messages, as shown in Figure 2-9.

Figure 2-9 Messages Window



```
Error: Application 'CreditHistory' -  Property 'Corporate Directory' (tm
Warning: Application Group 'admin' -  The members list is empty.
Warning: Application Group 'trusted users' -  The members list is e
Warning: Form Layout 'refused.html' -  There are no fields in the thi:
0 infos, 3 warnings, 1 errors.
```

You must fix all errors before you can deploy an application. You can deploy an application that contains warnings, but you may encounter difficulties later when you try to use the application.

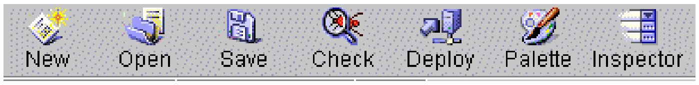
The last line of the window summarizes the information found by the syntax check.

If you right-click in the Messages window, a dialog box appears. You can use this dialog box to selectively display different types of messages (for example, errors only or warnings only). You can also clear the Messages window.

Main Toolbar

The main toolbar (see Figure 2-10) contains frequently-used commands.

Figure 2-10 Process Builder main toolbar



The toolbar contains the following buttons:

New Launches the New Application dialog box, where you create a new application which is not based on a previous application.

Open Opens an existing application.

Save Saves the application locally. Use this command to save your work as you are creating or editing an application.

Check Checks the application code and displays errors, warnings, and information messages in a separate window. You need to fix all errors in your application before you can deploy it. You can deploy an application for which you receive warnings, but it may lead to problems in the future. The last line of the window gives you a summary of how many of each type of message the check produced.

Deploy Deploys the application to a cluster.

Palette Displays the palette.

Inspector Displays the Inspector window for the selected element. This window allows you to set the properties of any element. You can also double-click the element's name in the application tree view to open the Inspector window. Some elements have evaluation order Inspector windows as well as property Inspector windows. Typically, an evaluation order window allows you to reorder conditions or roles into a specific hierarchy or sequence.

Menu Commands

Several menus provide commands for use as you design a Process Manager application. The menus are Application, Edit, Insert, Format, Window, and Help.

Application

New Creates a new application. If you have an application open, this closes the current application after asking if you want to save it first, and launches a new application with a fresh tree view and an empty process map.

Open Takes you to the "Select an Application" dialog box, from which you can open an existing application. If an application is already open, Process Builder closes the application. If the open application has unsaved changes, you are first prompted to save the application.

Import from ZIP Imports an existing zipped application. If you have an application open, this closes the current application after asking if you want to save it first. Browse to the zip file and import it into the current Application folder.

Delete Takes you to the "Select an Application" dialog box, from which you can delete an application. If an application is already open, Process Builder closes the application. If the open application has unsaved changes, you are first prompted to save the application.

Save Saves the currently displayed application to your local machine.

Save As Saves the current application with a different name.

Import File Brings up a dialog box, from which you can copy a file into the application's directory. Enter the path name of the file to import, or use the Browse button to browse to it. Then specify where you want the file copied. The file is automatically copied to a specific location, depending on the file category you select. (See Table 2-2.) If a subdirectory doesn't exist, it will be created.

Table 2-2 Destination folders for imported files

File Category	Destination Folder
Top Folder	The same directory as the current application. For example: `builder/Applications/app_name`
Client Images	`images` subdirectory
Client JavaScript	`js` subdirectory
Server JavaScript	`WFScripts` subdirectory
Server JAR Files	`lib` subdirectory
Mail Notification Templates	`templates` subdirectory

Check Errors Checks the application code and displays errors, warnings, and information messages in a separate window. You need to fix all errors in your application before you can deploy it. You can deploy an application for which you receive warnings, but it may lead to problems in the future. The last line of the window gives you a summary of how many of each type of message the check produced.

Deploy Deploys the application to a cluster.

Save Process Map Image Saves the process map as an image file. Use this option when you want to show the whole process map outside of Process Builder. For example, by saving the process map as an image, you can print it or insert it into an HTML document. You can choose where to save the image, and you can choose whether to save it in JPEG, PNG or BMP format.

Exit Quits Process Builder. If you have unsaved changes in an application, it gives you the option of saving them.

Edit

Cut Cuts an HTML layout element from a form, or cuts text from a script.

Copy Copies an HTML layout element into a form, or copies text into a script.

Paste Pastes an HTML layout element into a form, or pastes text into a script.

Delete Deletes a single element such as an activity, a data field, or a form from the application tree view. You cannot delete folders from the application tree view, or default item such as predefined scripts and default users and groups. Also, if the application has been deployed in the production stage, you may not be able to delete certain items.

Insert

Group & Role Inserts a new group or role.

Data Field Inserts a new field.

Form Inserts a new HTML form.

Script Inserts a new script.

HTML Element Inserts the HTML element that you select from the associated submenu. For more information, see "Insert HTML Element" on page 178.

Format

These are standard HTML editing commands. Use them to edit HTML forms in the HTML editor.

Size Applies the font size you select to the text.

Style Formats text as bold, italic, underline, or strikethrough.

Remove All Styles Removes size and style formatting from text.

Heading Applies an HTML heading tag to the text.

List Applies an HTML list tag to the text. You can choose either a bulleted list or a numbered list.

Align Aligns text at the left, right, or center of the page.

Decrease Indent Decreases the amount of indenting on the text.

Increase Indent Increases the amount of indenting on the text.

Window

Select one of these menu items to display the window, or, if it is already displayed, bring it to the foreground.

Project Window Displays the application tree view.

Process Map Displays the process map.

Palette Window Displays the palette.

Messages Window Displays the error and informational messages produced when you check the application syntax using the Check button.

In addition, you can reorganize the windows with either of the following menu items:

Auto Arrange Windows Arranges windows to fit side-by-side within the main window. Four windows will be displayed: the application tree view, process map, Messages window, and the palette. The resulting arrangement is not updated when you resize the main Process Builder window.

Always Arrange Windows Similar to Auto Arrange Windows, except that the arrangement is always updated. Whenever you resize the main window, the individual component windows are resized to maintain their side-by-side arrangement.

Help

Help contents Accesses Process Builder full table of contents. Click a link to go to the topic you are interested in, or use the Index to find a topic. If you click the Contents button within the window, you can access the "Other Documentation" link, which gives you access to the Process Manager documentation set.

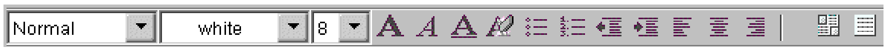
Custom Help Displays help you have customized for Process Builder at your site. For example, you might use this feature to add links to your Process Express site or to explain site-specific customizations to Process Builder. To add your own help, edit the file `custom_help.html`, found in the `support/html` directory of your Process Manager installation.

About Displays the software version information.

HTML Page Editor Toolbar

The HTML Page Editor toolbar allows you to format the HTML forms that you create. Figure 2-11 shows the HTML Page Editor toolbar.

Figure 2-11 HTML Page Editor toolbar



The drop-down lists and the icons assign HTML formatting to the text you select in your HTML forms. See "Using the HTML Page Editor Toolbar" on page 177 for more information.

Chapter 3

Planning an Application

This chapter describes the issues to consider when you are planning an application.

This chapter includes these sections:

Planning Overview

Process Builder provides a powerful and intuitive visual environment to create and deploy Process Manager applications. This chapter describes the things to think about before you develop your application. For an explanation of completed applications, see the sample application chapters in this book.

To plan your Process Manager application, you need to do the following tasks:

1. Decide on the sequence of steps involved in a process and whether you need to create subprocesses or custom activities.
2. Define who the people involved in the process are and what roles they play.

3. Decide whether you want to give people who aren't directly involved in the process access to some of the process information.
4. Decide what data you want the application to track on forms, internally within scripts, and what you need to pass to subprocesses and to other programs using custom activities.
5. Define a useful set of forms to allow users to complete each step, and if you like, to allow users to monitor the process.
6. Decide which users and groups should have access to which forms.
7. Decide if you need to write any special scripts for your application.
8. Decide if you want users to be able to search the application, and if so, for which information.

Planning the Process Map

Before you can create a Process Manager application, you must understand the process you want the application to handle.

The best way to create an application is to examine your current process, automated or manual, figure out the steps involved, and then improve the process with Process Manager's automation and database capabilities.

For more information on Process Maps, see Chapter 5, "Designing a Process Map."

Entry Points

After analyzing the existing process, you should decide at what point(s) in the process you want to start your application. You can have more than one entry point, depending upon the process, but you need to find starting points that still let you capture all the data you need to capture. If you start halfway through a process, you may not have all the data you need to carry the process through.

For example, if you have a process for posting documents on a web site, you might have one entry point for documents that need to be converted to HTML, and another entry point farther through the process for documents that are already in HTML.

For more information on entry points, see "Entry Points" on page 98.

User Activities

Each step in the process that a user needs to perform is called a user activity. When you figure out the steps in the process, you need to decide which steps need to be performed by a person and which can be performed automatically.

For example, in the application that handles an employee’s vacation request, you might have three activities:

- Manager approval, where the manager either approves the request or asks for clarification.
- Clarification, where the employee provides any requested clarification
- HR approval, where HR approves the request

Since these all require a user to take action, they are all user activities, not automated activities. For more information, see “User Activities” on page 99.

Automated Activities

If a step can happen automatically, you’ll want to create an automated activity, instead of an activity that requires a user to perform an action.

For example, the Data Sheet sample application uses an automated activity to build the data sheet. The automated activity runs a script that looks up price information and creates an HTML form based on this information. This saves a user from having to enter the data. The next step of the process is an activity that presents a user with the data for approval.

For more information on automated activities, see “Automated Activities” on page 105.

Custom Activities

You need to determine if you want to create a custom activity to connect to external services, to integrate information from other services into your Process Manager application, or to extend Process Manger to other services. For example, you can use custom activities to connect to iPlanet Application Server Extensions for CICS or SAP. A developer needs to write a Java class, write an XML file that describes the Java class, and put the two together in a .zip or .jar file. You need to decide what data you need to transfer to the external component, and what data you need to receive back.

For more information on custom activities, see "Custom Activities" on page 111.

Subprocesses

You need to determine if you want to create a subprocess that can be used across your environment. A subprocess is a fully functional Process Manager application that can be called from within another Process Manager application. The Loan Management sample application, which controls the process of approving a loan, calls the subprocess Credit History. Credit History can be run either as a subprocess of the Loan Management application or as a stand-alone application. For more information on subprocesses, see "Subprocesses" on page 107.

Decision Points

You need to determine if the process has a single set of steps, or if it branches because of conditions. Decision points are where the process branches. For example, in the Data Sheet sample application, there is a decision point where if the price of the product on the data sheet is less than $1000, the data sheet is immediately published. However, if the price is greater than or equal to $1000, the data sheet must be approved by the Vice President before being published.

For more information about decision points, see "Decision Points" on page 118.

Parallel Processing

You need to determine if you want to use parallel processing, which allows your process to have two or more branches so that two or more activities can execute in parallel. In the Office Setup sample application, which controls the process of setting up an office for a new employee, each subtask is grouped into a processing branch that progresses independently of the other subtasks. For example, the MIS department can set up the phone while the purchasing department is ordering the computer. Problems completing one task won't affect the progress of a parallel task. In complex processes, there can be several levels of nested parallel branches.

For more information about parallel processing, see "Split-Join (Parallel Processing)" on page 120.

Exit Points

You also need to decide where to end the process. You may have more than one exit point. For example, in the Time Off Request sample application, an employee's request for vacation time can end three ways: the vacation is approved or denied by management, or cancelled by the employee. Those three outcomes translate into three exit points in the application.

For more information on exit points, see "Exit Points" on page 124.

Transitions

Once you have the basic steps of the process in place, you need to determine the flow between them. The connections between steps are called transitions, and are represented on the process map by arrows connecting the steps. Since a decision point requires the process to branch, each branch has a transition leading to a different step.

For more information on transitions, see "Transitions" on page 124.

Notifications

Once you have the steps and transitions in place, you need to think about where in the process you want to set up email notifications, what information the notification contains, and who needs to receive the notifications. The notifications are sent as soon as the process reaches the step.

For example, in the Time Off Request sample application, the employee is notified if the vacation is approved or denied at the exit points for approved vacation and denied vacation. At the exit point where the employee cancels the vacation, no notification is required.

For more information on notifications, see "Notifications" on page 122.

Exceptions

Exception handling allows the administrator to intervene manually if errors occur. Every process step (other than an entry point) must be assigned an Exception. See "Exception Manager" on page 116 for more information on exception handling.

Planning Assignments

You need to look at each activity and determine who needs to do it. The person who performs an activity is called the assignee. You determine an activity's assignee by asking the following questions:

- Who needs to perform this step?
- Should this step be assigned to a single individual, or to a group of individuals so that any one of the group can perform the work item?
- Is this an activity where a number of individuals need to approve some information before the process continues (parallel approval)?
- Does the person who performs the activity vary depending upon the process instance? For example, a vacation request might have an activity for the manager's approval. The manager who needs to approve the vacation, depends on which employee requested the vacation for a particular process instance. In other cases, the assignee might be the same regardless of the process instance. For example, if every purchase over a certain amount needs to be approved by the Chief Financial Officer, the assignee of that approval activity is the same regardless of the process instance.
- If the assignee depends upon the particular process instance, what information do you need to have in order to determine who the assignee should be?

Once you have the answers to these questions, you know who needs to be assigned to activities, and that information helps you set up the groups and roles you need for an application.

For more information on assignments, see "Setting Activity Assignments" on page 102.

Planning Groups and Roles

Once you have figured out who needs to participate in the process and what they need to do in the process, you need to think about your groups and roles. If a user activity can be performed by any one person in a group, you must set up a group to assign the activity to.

For example, in the Office Setup sample application, one user activity is "Order Computer." Anybody in the purchasing group can perform this step, so the activity is assigned to a group called Purchasing.

If you have an assignee that varies by process instance, and the information needed to identify this person is stored in a data field for each process instance, you can set up a field role. For example, in a process where a web designer is working with a graphic artist on a web page, you could create a field role for the artist. The form initiating the web page art could have a field on it where the designer fills in the artist's name. The application then stores the artist for a particular process instance, and uses it throughout the process.

To figure out what groups and roles you need, ask the following questions:

- Who in the company can initiate a process?
- If approval is needed, what needs approval and who needs to do it?
- If you need to use a group of people in your application, is the group already defined in the corporate user directory?

There are four default groups and roles for an application:

- creator (the person who initiates a process instance)
- admin (people allowed to administer the application)
- all (any interested party)
- trusted users (people allowed to connect to the subprocess)

If you need additional groups and roles, you need to create them. If the group already exists in your corporate user directory, you can create a group for your application that is tied to the group in the corporate user directory.

For more information on groups and roles, see Chapter 6, "Defining Groups and Roles."

Planning for Monitoring

In addition to the assignees in an application, you can also let other people who aren't directly involved in the application monitor the progress or view information on a read-only basis. If you want to give these nonparticipants access to information, you may need to create additional groups and roles.

Planning Delegations

For some activities, you may want the assignee to be able to delegate the work item to another user. If you want the assignee to be able to delegate, you must set the activity's Allow Delegation property to yes. Before allowing delegation, though, you need to think about whether the activity should be delegated. For example, if you had a process for approving departmental salary increases, you may want the head of the department to approve the increases personally. In this example, you would not allow delegation.

Determining the Data Requirements

After you have defined the steps in your process, you need to define what data you need to track. Once you've determined the data you need, then you can create the process' data dictionary using Process Builder. The following questions help you determine what data fields you need:

- What information do assignees need to see to perform each step?
- What information do you need to gather from assignees at each step?
- What data do you want to have a record of?
- What information do the scripts in the application require?
- If you're using subprocesses, what information is passed between a parent and child process?
- If you're using a custom activity, what information is passed between the Process Manager application and the external component or service?

For each data field you create, you need to think about how to present it to users on the form (for example, is it a text field, radio buttons, or a file users need to attach?). You also need to think about how it will be stored in the database (is the data type text, a date, or an integer?)

For more information on data fields, see Chapter 7, "Defining Data Fields."

If the standard data field classes provided with Process Manager do not fit your needs, you can create your own. For example, you can use custom data fields to access external data sources and to generate dynamic content at entry points. To use custom data fields you need to write Java classes that implement the field.

File Attachments

Some information your application gathers may be files users contribute by attaching them to forms. You create specific data fields for file attachments. If you are using file attachment data fields in your application, you must plan for their use:

- You must define a URL on your Enterprise Server to store these documents.
- You must set the Enterprise Server's access privileges for the URL, so that the documents are available to a public user.

  Setting up the public user allows all users to upload documents to that area. Web Publishing must be turned on in your Enterprise Server.

For more information on file attachments and the Enterprise Server, see "Setting Up the Content Store" on page 168.

Planning Forms

Once you have thought through the process steps, the data requirements, and the people involved you need to think about the forms to use to collect the data, move it from person to person, and display the data to people involved in the process.

You can design a different form for each role or group at each step in the process. Different forms let you display only the information needed by a particular person or group. For example, the assignee of an activity often has a special form which has more information on it than the forms for other people involved in the process, or people who are monitoring the process. Often the creator of the process instance has forms so he or she can follow the process instance's progress.

You create forms in a form wizard in Process Builder. You add fields to the forms in a form editor by dragging them from the data dictionary to the form. You need to decide which fields to put on a form, and whether they can be edited by the assignee or not.

For more information on forms, see Chapter 8, "Designing Forms."

Planning Access to Forms

Using form access you control which users see which forms. To plan form access for a process, use the following questions:

- At what step in the process is the form used?
- Who views the form (is the viewer the assignee, a participant in the process, or an observer?) If the viewer is a participant in the process, what group or role do they belong to?
- Do you want the form to be viewed by multiple groups and roles?

When you have the answers to these questions, you create the association between an activity, the role or group, and the form in the Form Access window.

For more information on form access, see "Setting Access to Forms" on page 182.

Planning Custom Scripts

Many of an application's actions are performed by JavaScript scripts. For example, JavaScript assignment scripts determine which user is assigned to a step. Process Builder includes some standard scripts; however you may find you need to write additional JavaScript scripts to perform actions specific to your application. In planning these scripts, consider what kind of scripts you need to write and whether you already have similar scripts you can base them on. Scripts you plan to reuse or plan to call from other scripts are stored in the Toolkit folder.

For more information on using scripts, see Chapter 9, "Using Scripts."

Planning Searches

If you want users to be able to search for information about process instances, or for specific field information, you need to design your application so that searching is allowed. Use the following questions to help you plan your searches:

- Who do I want to allow to search application data?
- What fields do I want to allow people to use for searching? For example, in an application for publishing documents to a web site, you might want to be able to search based on the author's name or the document title.

For more information on searching, see Chapter 13, "Setting Up Searching."

Planning Deployment

After you have completed the application, you may want to deploy it for testing before rolling it out to a production server. In test mode, the person who designed the application can do a walkthrough to see that all steps are correct and that the information flows through the system correctly. In test mode, all activities are assigned to the creator of a process instance, so you cannot test form access fully until you deploy in non-test mode. Only then does the application use all the groups and roles.

Process Manager has two deployment stages: development and production. When you deploy to development, you can still change everything about an application. However, once you deploy to production, some information about the application can no longer be changed.

After you have tested the application and made any necessary fixes, you can deploy it for use in real-world environments. Once the application is deployed to end users, it usually leaves the builder's control. However, even after end users are using it, the builder and the process administrator need to communicate whether the application needs changes, if it needs to be replaced, or if it is made obsolete.

For more information on deploying applications and editing deployed applications, see Chapter 10, "Deploying an Application."

Chapter 4

Creating an Application

This chapter takes you through the process of creating a new application. First it outlines the process.

This chapter includes these sections:

Application Creation Overview

After planning your process application, you're ready to create it. Here are the main steps:

1. Get information from the Process Manager administrator.

   See "Getting Information from the Administrator" on page 84 for more information.

2. Launch Process Builder and create a new application.

   See "Creating a New Application" on page 85 for more information.

3. Design a process map, including all activities that users need to perform.

   See Chapter 5, "Designing a Process Map" for more information.

4. Define the groups and user roles for process steps participants and non-participants (observers).

   See Chapter 6, "Defining Groups and Roles" for more information.

5. Define the assignees for a step.

   See "Setting Activity Assignments" on page 102 for more information.

6. Define the data fields, and set up the file attachments you need.

   See Chapter 7, "Defining Data Fields" for more information.

7. Design the forms to display the appropriate data fields and to attach the necessary files.

   See Chapter 8, "Designing Forms" for more information.

8. Set the form access, thereby associating the form with the appropriate step and with the appropriate participants for that step.

   See Chapter 8, "Designing Forms" for more information.

9. Decide which users, if any, are allowed to perform searches. Also decide which fields are searchable.

   See Chapter 13, "Setting Up Searching" for more information.

Getting Information from the Administrator

In order to successfully develop and deploy an application, you need to work with the Process Manager administrator to set certain values for the application. In general, information dealing with clusters, the Enterprise Server, and the database may be set by your administrator. For example you may need to work with your administrator to set appropriate values for the following fields:

- the application properties DB User and DB Password, the database user and password which has the privileges to create database tables
- the content store properties URL, Public User and Public Password. These properties are the URLs on the Enterprise Server to which the attached files are posted, and the user and password for accessing the URL
- the person or people allowed to administer the application (the people you add to the "admin" group)
- the Process Manager administration user and password needed when you deploy the application to a cluster

For more information on these values, see “Setting Application Properties” on page 87, “Setting Up the Content Store” on page 168, “Default Groups and Roles” on page 131, and “The Deploy Application Dialog Box” on page 208.

Creating a New Application

Since each process is handled by its own application, the first thing you need to do after planning the process is to create a new Process Manager application for it.

➤ **To create a new application**

1. Launch Process Builder.
2. In the Select an Application dialog box, click New Application.
3. Fill out the dialog box fields.

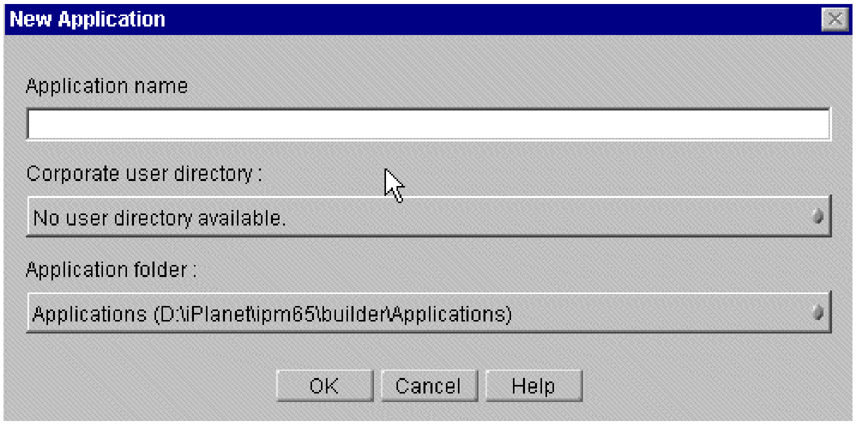
   See “The New Application Dialog Box” on page 85 for details.
4. Click OK. This displays a default set of empty resource folders and a blank process map.

   See Chapter 5, “Designing a Process Map” for details.
5. Set the application’s properties in the Inspector window.

The New Application Dialog Box

The New Application dialog box (see Figure 4-1) is where you enter the name and location of a new application. The location you enter here is the location on your local system. Typically you have a “working” copy of the application on your local system. Once you deploy an application, you generally use the deployed application as the basis of any modifications you want to make.

Figure 4-1 The New Application dialog box


New Application
Application name
Corporate user directory :
No user directory available.
Application folder :
Applications (D:\iPlanet\ipm65\builder\Applications)
OK
Cancel
Help

Application name Enter a name for your application.The field is required. It can contain only the following characters: A to Z; a to z; 0 to 9 and _ (underscore). All other characters are not allowed.

Corporate user directory This directory contains all valid users and groups for the corporation, and is the source of the user information you need in order to set up user groups and roles for the application. The available directories that appear in the drop-down list are defined in the `preferences.ini` file.

Application folder This is the full physical path to where you want to save this application on your local machine. By default the path is the main applications folder on your local machine, *NAS_Root*`/builder/Applications`. The following folders are available from the drop-down list:

- Applications: The main folder for storing applications.
- Application Templates: The folder where you store applications that you base other applications on. For example, if your applications share many elements in common, you might want to designate a template application, save it to this folder, and use it as a basis for future applications. A sample template application, Default, is included in this folder as an example.
- User Applications: A folder for applications stored by user. On Unix, this folder is your default user home directory.

- Sample Applications: A folder that contains all the applications shipped as samples with Process Builder. This folder is defined in your `preferences.ini` file.
- Scratch Folder: A folder for storing temporary, or scratch, versions of your applications. This folder is defined in your `preferences.ini` file.

Setting Application Properties

The application has properties that you need to set. These properties affect the application as a whole.

➤ **To set the application properties**

1. Double-click the application's name at the top of the application tree view or select the application's name and click the Inspector icon, or right-click the application's name and choose Properties.
2. Fill in the fields.

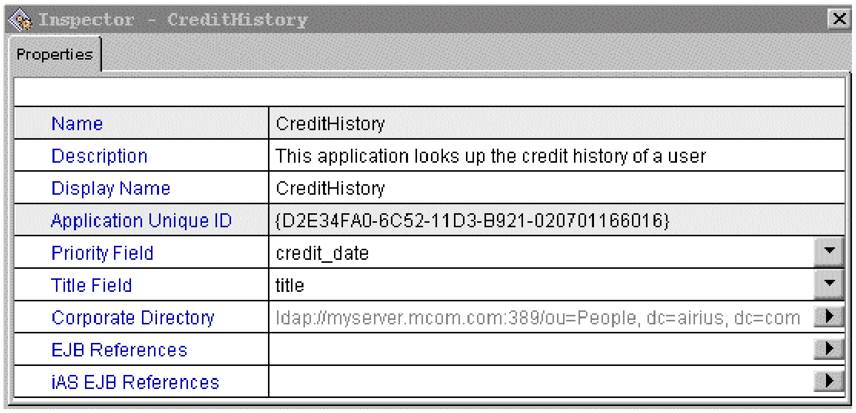
   See "The Application Properties Dialog Box" on page 87.
3. When you're done, close the window by clicking the close box.

   The changes you make are saved automatically.

The Application Properties Dialog Box

The Application Properties dialog box (see Figure 4-2) includes the properties you set for the application as a whole. Note that you cannot change fields that are grayed out. The fields that are grayed out vary depending on whether you deployed your application in the Development or Production stage. For more information on deployment, see Chapter 10, "Deploying an Application."

Figure 4-2 The Application Properties dialog box



Name	CreditHistory
Description	This application looks up the credit history of a user
Display Name	CreditHistory
Application Unique ID	{D2E34FA0-6C52-11D3-B921-020701166016}
Priority Field	credit_date
Title Field	title
Corporate Directory	ldap://myserver.mcom.com:389/ou=People, dc=airius, dc=com
EJB References	
iAS EJB References	

Name The name of the application that appears in the list of applications. This field is required.

Description A longer description of the application. This description appears in Process Express. The user has to have enough information between the Display Name and the Description to identify the application they want from a list of applications, so it's important to make these two fields descriptive.

Display Name The longer, more descriptive name displayed by Process Express. If you are updating an existing application by saving it with a new application name, you can still keep the same Display Name, so that the name users see can be the same from version to version of the application.

Application Unique ID An internal unique ID generated automatically to track each application. This ID identifies the application, and differentiates applications with the same name. If you do a "Save As" on an application, you can save the application using the same name, but the unique ID will be different. If you attempt to deploy an application that has the same name as an existing deployed application, but a different unique ID, you are prevented from deploying.

Priority Field To set up your application so that process instances can be prioritized, use this property. First you need to create a data field of class ID Select List to represent the priority. The field must have the value of an integer between 1 and 5 (1 is the highest priority; 5 is the lowest). Next you choose this field to be your priority field using this property. Then you put the priority field in a form so that the person responsible for prioritizing work items can select a priority value for the application's work items. If you do not use it, all process instances have a priority of 3 (normal). The behavior of your application is not affected by the value of this field.

Title Field The data field that contains the title of a process instance. This field is required. This field appears in Process Express as the title of the process instance, so you need to set this property to a field that has a value that describes the process instance and that end users will find helpful. This field will also be the title of any email notifications sent. It is often best to create a special field to be the title field. The field must already exist before you can fill it in here. This field is required.

Corporate Directory The corporate LDAP directory this application uses for user and group information. A pop-up window lets you choose a directory, either based on a cluster or by name. The listed directories are derived from the corporate user directory values in your `preferences.ini` file. The directory you choose must be the same as the directory of the cluster to which you will deploy the application. If you choose the directory based on a cluster, your directory will be up-to-date, even if the administrator changes the directory for the cluster. However, if the administrator changes the directory for the cluster, you have to make sure your users and groups correspond in both directories.

Using Sample Applications

Process Manager ships with sample applications that demonstrate how to build applications, as well as showing the kinds of tasks you can perform in an application. The available samples are listed in Table 4-1.

Table 4-1 Sample Applications

Application	Description
DataSheet	Creates a product data sheet, including setting pricing and getting signed approval by two levels of management. For more information, see Chapter 14, "The Data Sheet Application."
OfficeSetup	Sets up an office for a new employee, including assigning an office space, ordering a computer, and installing the phone and computer. For more information, see Chapter 15, "The Office Setup Application."
LoanMgmt	Submits an application for a loan, including launching a subprocess (CreditHistory) to check the applicant's credit history. For more information, see Chapter 16, "The Loan Management and Credit History Applications."
CreditHistory	Checks a loan applicant's previous credit history with the company. Used as a subprocess by the LoanMgmt application. For more information, see Chapter 16, "The Loan Management and Credit History Applications."

Table 4-1 Sample Applications *(Continued)*

Application	Description
ClaimProcess	Submits an insurance claim, including checking policy details from a flat file, and going through the approval process at the insurance company. For more information, see Chapter 17, "The Insurance Claim Processing Application."
HelloWorld	Demonstrates how to use a custom activity in an application. The custom activity displays a greeting based on the language specified for the greeting. For more information, see the Custom Activities chapter in the *Process Manager Programmer's Guide*.
TimeOffRequest	Submits an employee's request for time off for approval by the manager and by human resources (HR).

You can use these applications as models for your own application. To base an application on one of these existing applications, use Save As and make sure your new application has a unique name and a unique database table.

After configuring a sample application, you can deploy it in development mode to your own cluster. In this way, you can see how applications are managed in Process Administrator. You can also see how end users run them in Process Express.

Applications and the Corporate Directory

When you install Process Manager, the default installation uses the same Directory Server for both the configuration directory and the corporate user directory. This user directory contains a small set of sample users, along with users defined as administrative users during the Process Manager installation. For example, if you define `admin` as the administrative user, the installer adds this to the sample set of users in the directory.

If you do a default installation, using the `admin` user, you can easily set up the sample applications to deploy.

If you do a custom install or otherwise change your configuration, you may need to make adjustments to the samples before you can deploy them. For example, if you defined a different user directory (such as your company's own corporate directory) or if you want to use other users, you must adjust the sample applications.

Most of the sample applications define `admin` as their default user. If you use your company's corporate directory, then you must do either of the following before deploying the sample applications:

- Make sure your company's corporate directory contains an `admin` user.
- Use another user ID, such as your own or a test user ID.

If you change your corporate directory after your cluster has been created, there are several changes that you and the server administrator must make before you can deploy applications with the new directory. For information on changing your user directory, see the *Process Administrator's and Business Manager's Guide*.

Setting Your Corporate Directory

Before you can add or change the default users for the sample applications, you must define a corporate directory for each application. Otherwise, you cannot browse or search the directory of users. To define a directory, perform the following steps:

1. Open the application.
2. In the application tree view, highlight the name of the application, right-click, and choose Properties.
3. Click the right arrow in the Corporate Directory property field. A selection dialog box appears for the corporate user directory.
4. Choose whether to set the directory based on the cluster or not.
    - If you have a cluster available during the design phase, choose the cluster-based directory, and select the cluster you want.
    - If you don't have access to a cluster, choose to point to a specific directory. A drop-down list appears, listing all corporate directories identified in the `preferences.ini` file. Note that if you point to a specific directory, you may need to change this value when you deploy the application. This will be necessary if your deployment cluster uses a corporate directory different from the one you chose.
5. Click OK.

Deleting an Application

You can delete applications only when they are stored on your local machine. To delete an application, perform the following steps:

1. In the Select an Application window, select an application.

   If this window is not open, you can open it as follows. From the Application menu, choose Delete.

2. In the Select an Application window, click the Delete button.

   A confirmation dialog box appears.

3. Click Yes to delete the application.

Chapter 5

Designing a Process Map

This chapter describes the elements of a process map, their properties, and how to use them to design a process map.

This chapter includes the following sections:

Drawing the Process Map

When you create a new application, Process Builder displays an initial set of empty resource folders and a blank process map. As you draw the map, Process Builder adds items to the folders in the application tree view.

➤ **To design a process map**

1. Create a new application.
2. Drag items from the Palette to the Process Map pane.
3. Update the properties of the items you add to the process map.
4. Add transitions between the items.

A process map is the visual representation of an application's steps. You also need to create all the other items, forms, data fields, and so on, in order to make a working application.

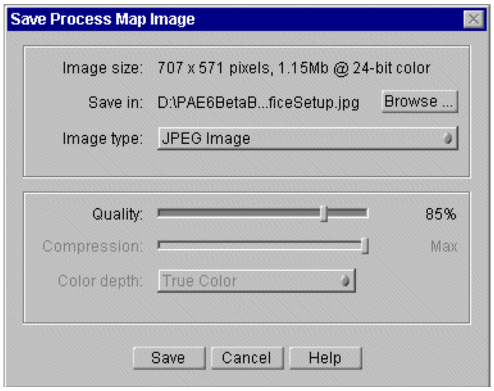
Saving a Process Map to a File

Sometimes you want to export the process map—in other words, save the map outside of Process Builder. For example, by saving the process map as an image, you can later print it or insert it into an HTML document.

To export the process map, open the Application menu and choose "Save Process Map Image."

The "Save Process Map Image" dialog box appears, as shown in Figure 5-1. This dialog box lets you choose what kind of image file to save the process map as.

Figure 5-1 Save Process Map Image Dialog box



To choose where to save the image, select the Browse button, and navigate to the desired folder.

To choose what kind of image file to save the process map as, click the Image Type field, and choose an image type from the menu. The choices are:

- JPEG Image

  This encodes and compresses the file and saves it with some lossage (that is, it drops some pixels). This format is great for portability and inclusion in web pages but occasionally you will see color bleeding. This format usually gives the smallest file size.

  This option lets you select the quality of the image as a percentage (0 is lowest quality, 100 is highest quality).

- BMP Windows Bitmap

  This saves the image as a straight RGB image with no encoding, compression, or lossage. Use this format if you want an exact replica of your process map.

  This option allows you to select the color depth of the image.

- PNG Image

  This saves the image in the portable network graphics format, which is a compressed, possibly color indexed image, with no lossage. PNG format was designed as a replacement for GIF due to copyright issues with GIF. It generally results in good compression.

  This option allows you to select amount of compression and the color depth of the image.

After the process map is saved, a dialog box appears, as shown in Figure 5-2. You can click OK to close this dialog box, or you can click the image link (for example, "JPEG image") to display the new image file that was created.

Figure 5-2 Select the image link to view your saved process map

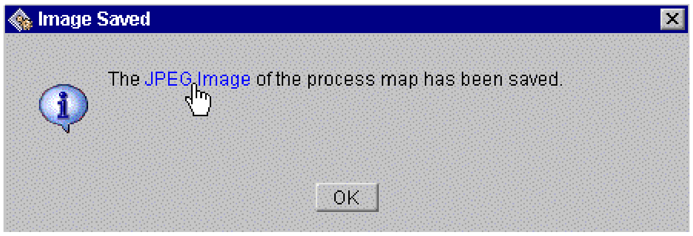


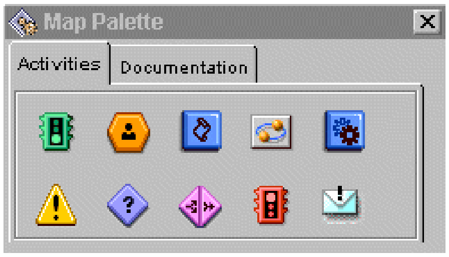
Image Saved

The JPEG Image of the process map has been saved.

Adding Items with the Map Palette

You create a process map by dragging items from the Map Palette (shown in Figure 5-3) to the process map.

Figure 5-3 The map palette



Using the Map Palette, you can add the map items described in Table 5-1.

Table 5-1 Icons in the Activities tab

Icon	Description
🚦	**Entry Point.** A point at which a user can initiate a process. An application can have several entry points. For example, if the first few steps create an ID number for the user, a returning user who already has an ID can skip those steps.
⬢	**User Activity.** A step within the process that requires a user to perform a task. Each user activity has an assigned user who performs the task (assignee) and a form the user needs to fill out in Process Express. After you place activities in the process map, you define the sequence in which they are to be executed by connecting them with transition lines.
🟦	**Automated Activity.** An automated step performed through a JavaScript script without user intervention.
▭	**Subprocess.** A step that calls a process from within another process. The process that calls the subprocess is considered to be the main, or parent process, and the subprocess is considered to be its subordinate or child process. A parent process can have several children processes, each of which is a stand-alone process complete with entry and exit points.
⚠	**Exception Manager.** A step that allows the administrator to intervene manually if errors occur when users run the application.
◆	**Decision Point.** A conditional step that causes the process to use different steps based on a condition. For example, you might have a decision point that directs the process to different steps depending upon the cost of an item.
◆	**Split-Join (parallel processing).** A step within the process that branches in two or more branches so that two or more activities can execute in parallel.
🚦	**Exit Point.** A step at which the process ends. An application can have several exit points. For example, in a vacation time off request application, an exit point could be the approved vacation request, and another could be a vacation that was not approved.
⚙	**Custom Activity.** A step at which a Process Manager application connects to external components or services.
✉	**Notification.** An email notification that is triggered when a user activity is started. The email can serve many purposes. For example, it can inform the person who started the process or other users of the process's progress.

➤ **To add items to your process map from the Palette**

1. Click the Palette icon in the strip below the menu bar to display the Palette, if it's not displayed already.
2. Drag an item from the palette to the process map.

   For notifications, you must drag the item into the user activity or exit point that already exists on the process map.
3. Double-click the item or highlight it and click the Inspector icon.
4. In the Inspector dialog box, define the item's properties.

   You can also right-click the items in the process map or application tree view. Since notifications do not appear on the process map, you must select them from the application tree view. For more information on the properties of specific items, see the section in this chapter for the type of item you added.

Deleting Items

To delete a process map item, you can do one of the following in the application tree view or the process map:

- highlight the item and press the Delete key
- right click the item and choose Delete from the menu
- highlight the item from the Edit menu and choose Delete

When you delete an item that is connected to other items by transitions (actions), you must also delete the transitions. If you attempt to delete an item with transitions, a dialog box appears asking if you also want to delete the transitions.

Entry Points

An entry point is a point at which a user can create a process instance. You can have more than one entry point; for example, if you are designing an application to create a data sheet, you might have two entry points. One entry point might be for data sheets for which you have to create art. Another entry point, later in the process, might be for data sheets for which you already have art prepared.

Entry points have the following properties:

Name The name of the entry point that appears in Process Builder and Process Express.

Description An optional longer description of the entry point that appears in Process Express.

Completion Script A script that runs when the step is completed. You must have already written the completion script you want before filling in this property.

User Activities

A user activity is a task that must be performed by a participant (the assignee) as part of a process. For example, a manager may have to approve an employee's time off. A form associated with the activity enables the assignee to take the required action so that the process moves to the next step.

User activities have the following properties:

Name The name of the activity that appears in Process Builder, and in Process Express. Since this name appears on users' work lists, make it descriptive of the action they need to take.

Description An optional longer description of the activity that appears in Process Express.

Allow to Save True/false. When set to true, this option allows users to save a work item so that they can complete it at a later time. If you set this property to true, a Save button appears on the work item's form in Process Express. Defaults to false

Allow to Add Comment True/false. When you set this option to true, end users are allowed to add comments to the activity. The comment field appears automatically on forms presented to users for that activity in Process Express. The assignee can add comments, which are then displayed in Process Express as part of the process instance's history. Defaults to true.

Allow to Delegate True/false. When set to true this option allows users to delegate a step so that someone else can handle it for them. If a step is delegated, the person it is delegated to becomes the step assignee, with the associated access to forms. If you set this property to true, a "Delegate" button appears on the work item's form in Process Express. Defaults to false.

Assignment Script The script that assigns a user or group of users to a work item. When a user is assigned a work item, it appears in the user's work list in Process Express. See "Setting Activity Assignments" on page 102 for further details about the script properties. You can use a predefined assignment script or create your own. If you create your own, the script must already exist before you can enter it in this property.

Expiration Setter Script The script that sets an expiration date or time for a work item. See "Setting Activity Expirations" on page 100 for further details about the script properties. The script must already exist before you can enter it in this property.

Expiration Handler Script The script that runs when a work item has not been completed by its expiration date or time. The script must already exist before you can enter it in this property.

Verification Script A script that runs when the step is completed. The verification script must already exist before you can enter it in this property.

Completion Script A script that runs when the step is completed. The completion script must already exist before you can enter it in this property. The completion script runs after the verification script.

Exception Manager If an error occurs, the Exception Manager invokes a new work item for correcting the error. Errors can occur in two cases: if a verification or completion script returns false, or if a runtime error occurs in a script (for example, in an assignment, expiration setter, or expiration handler script). See "Exception Manager" on page 116.

Setting Activity Expirations

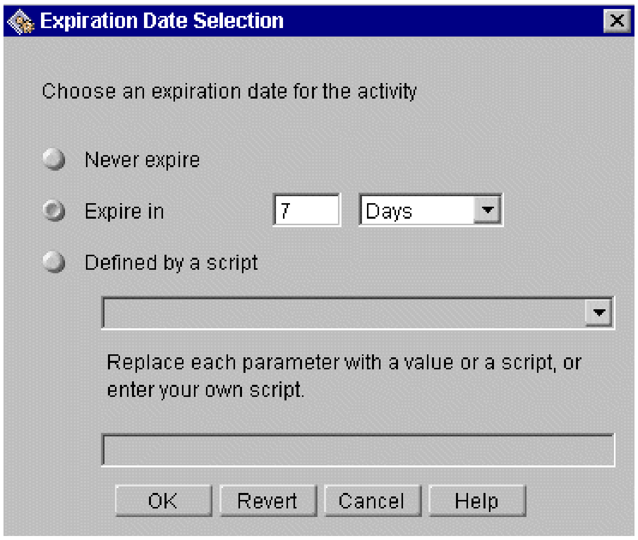
When you create a user activity, you can set an expiration date or time for it in the Expiration Date Selection window, either by specifying an expiration or a script to set the expiration.

➤ **To set an expiration**

1. Open the Inspector Window for an activity by double-clicking the activity or highlighting it and clicking the Inspector icon, or right-click and choose Properties.
2. From the Expiration Setter Script field, click the right arrow button.

   The Expiration Date Selection dialog box appears, as shown in Figure 5-4.

Figure 5-4 The Expiration Date Selection dialog box


Expiration Date Selection
Choose an expiration date for the activity
Never expire
Expire in
7
Days
Defined by a script
Replace each parameter with a value or a script, or enter your own script.
OK
Revert
Cancel
Help

3. Select the radio button for the type of expiration you want.
4. Enter expiration options for the activity.
5. Select OK to register your expiration date options and close the dialog box.

The following options are available:

Never expire The activity never expires. This is the default value.

Expire in The activity expires in a specified amount of time. You set the number and a unit of measurement (Months, Weeks, Days, Hours, Minutes).

Defined by a script The script that determines when the activity expires. The product contains no built-in scripts of this type. You can write your own JavaScript scripts, which will then be available in the drop-down list below the Defined by a script radio button.

If you need to set parameters for your script, you define them in the last field in the dialog box. You can give the parameter values, or the names of scripts that return the value. You can also enter your own script here. Enter your own script if you want to use a script that you haven't written yet, or if you want to use a script that is not a expiration setter script (for example, a toolkit script). If you write your own script, it must follow the conventions for expiration setter scripts.

If you set an expiration time or date, you can also set an Expire Manager Script for the activity. An Expire Manager script runs when the activity expires. This script is optional. You must create your own JavaScript script, since none of these scripts are built-in.

Setting Activity Assignments

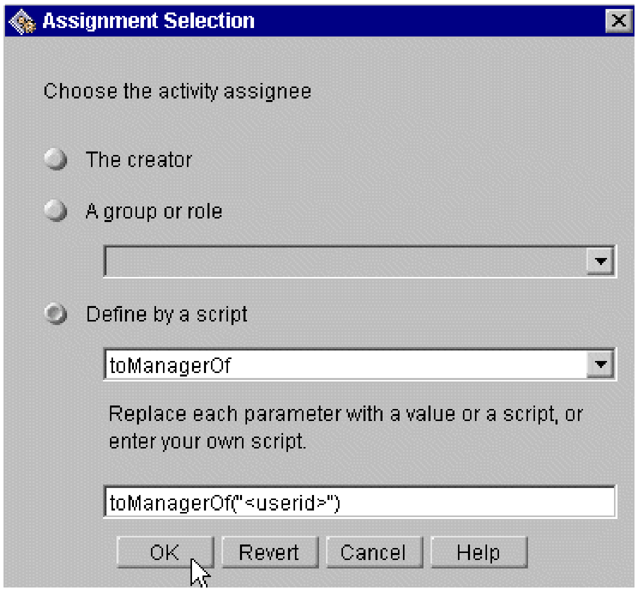
You can set a user to be assigned to an activity through an assignment script, many of which are built into Process Builder. You may need to set up some groups and roles before you can assign activities. See Chapter 6, "Defining Groups and Roles" for more information.

➤ **To assign an activity**

1. Right-click the activity in the process map or application tree view and choose "Set assignee script". You can also access the property page and click the Assignment Script field.

   The Assignment Selection dialog box appears, as shown in Figure 5-5.

Figure 5-5 The Assignment Selection dialog box



Choose the activity assignee

The creator

A group or role

Define by a script

toManagerOf

Replace each parameter with a value or a script, or enter your own script.

toManagerOf("<userid>")

2. Select the radio button beside the type of assignment you want.
3. Enter any assignment information in the fields.
4. Select OK to register your assignment script options and close the dialog box.

There are three ways to specify an assignment:

The creator Assigns this activity to the creator (the person who initiated the process instance).

A group or a role Select one of the roles or groups in the drop-down list. The list contains all the default roles and groups, as well as the ones you've created. If you assign a task to a group, the person in the group who first accepts the work item in Process Express performs the task. The other members of the group do not perform the task. If you want to assign a task to every member of a group for approval, see "Using Parallel Approval" on page 103.

Defined by a script Assigns this activity using a script—either a built-in script specified in the drop-down list, or a script whose name you specify in the text field below the radio button.

If you select the "Defined by a script" radio button, you must supply values for parameters contained in angle brackets. For example, using the assignment script `toManagerOfRole`, you are asked to complete the parameter in `toManagerOfRole("`*<rolename>*`")`. You must replace *<rolename>* with the field role: for example, `toManagerOfRole("product_mgr")`. You can also replace parameters with scripts that return the parameter value.

For details on the available assignment scripts you can specify, see "Assignment Scripts" on page 191.

Using Parallel Approval

Usually only one user is assigned an activity. However, in some cases you might want to have several users perform the same activity consecutively, for example, if you want several people to approve a document or action before it moves on to the next step. To implement an activity of this sort, you use parallel approval.

You implement parallel approval by creating an activity with specific assignment and completion scripts, and a data field to store necessary information.

➤ **To set up a parallel approval activity**

1. Create a data field *fieldname* to keep track of who has already performed the review and who still needs to perform it. This field must be a computed field with length of 2000.
2. Add an activity to the process map.
3. Right-click the activity in the application tree view and choose Properties to go to the Properties window for this activity (which will be your parallel approval activity).
4. Click the arrow on the Assignment Script property to bring up the window for setting the assignment script.
5. Choose `toParallelApproval` from the drop-down list.
6. Replace the parameters *arrayOfUserDNs* and *fieldname* with appropriate values.
7. Click OK to return to the activity's Properties window.
8. Click the arrow on the Completion Script property to bring up the window for setting the completion script.
9. From the drop-down list, choose `checkParallelApproval`.
10. Replace the parameters *fieldName* and *lableOfStopperAction* with appropriate values, as described in the next section.
11. Click OK.
12. Fill in any other properties you need to for the activity.
13. Close the window. Your changes are saved automatically.

From the end user point of view, a parallel approval item is assigned to all users in the group that need to give approval. If a user starts the work item and approves it, Process Manager stores the result in the data field and reassigns the work item to every user but the user who has approved the work item. This continues until all users have approved the item, or until one of the users has not approved the item. If all users have approved the item, the process continues in one direction. If one of the users has not approved the item, the process continues in another direction.

Parallel Approval Completion Script

To use parallel approval, you must use the completion script `checkParallelApproval(`*fieldName*, *labelOfStopperAction*`)`.

This script runs when the parallel approval activity is completed. If any user chooses the “stopper action” (that is, refuses to approve the item) the completion script performs the appropriate action. If all users complete the activity without choosing the “stopper action” (that is, all approve the item) this script performs the appropriate action.

The parameters are `fieldName` and `labelOfStopperAction`.

- The `fieldName` is the field that keeps track of who has performed the approval and who still needs to do so. The field is a computed field of length 2000 that you have to add to the data dictionary.
- The `labelOfStopperAction` is the name of the action or transition that a user can select that stops the approval.

Automated Activities

Automated activities are steps performed by Process Manager without user intervention, through one or more JavaScript scripts. Automated activities are triggered as soon as the process instance reaches the activity, unless the activity is deferred. If the activity is deferred, it is triggered at its specified date and time.

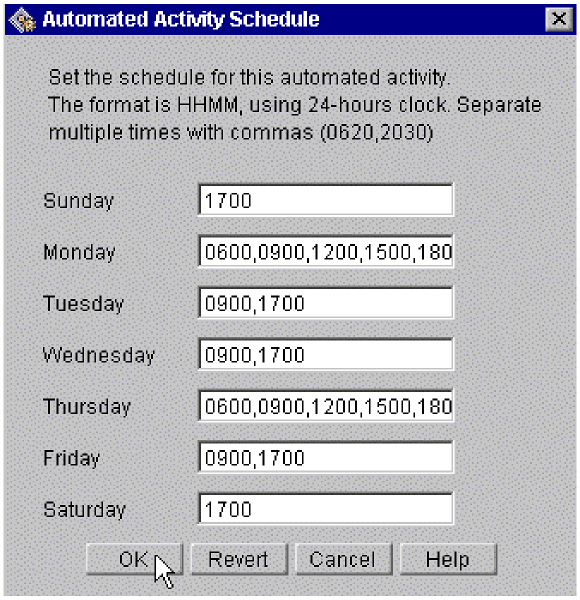
Automated activities have the following properties:

Name The name of the automated activity that appears in Process Builder.

Description An optional longer description of the activity.

Schedule The schedule for deferred activity. You need to set the Deferred property to true to use this schedule. You set the schedule property in a pop-up dialog box, shown in Figure 5-6.

Figure 5-6 Automated Activity Schedule dialog box



Automated Activity Schedule

Set the schedule for this automated activity.
The format is HHMM, using 24-hours clock. Separate multiple times with commas (0620,2030)

Sunday	1700
Monday	0600,0900,1200,1500,180
Tuesday	0900,1700
Wednesday	0900,1700
Thursday	0600,0900,1200,1500,180
Friday	0900,1700
Saturday	1700

Enter a time into any of the day fields to have the automated activity run at that time on that date. Enter times in the 24-hour clock format (for example, 1300 for 1:00 pm To enter more than one time for a day, separate the times with a comma. If you enter only two digits, Process Builder assumes they are the hour.

Deferred This property specifies if the activity should be performed as soon as the process reaches the automated activity, or if it should be deferred. If you set the property to true, the automated activity is deferred to the time you specify in the Schedule property. If this property is set to False, the automated activity is performed immediately. The default is false.

Automation Script The script that performs the automated step. The script must already exist before you can enter it in this property.

Verification Script A script that runs when the step is completed. The script must already exist before you can enter it in this property. For automated activities, the step is completed when the automation script has finished running, so the verification script runs immediately after the automation script.

Completion Script A script that runs when the step is completed. The script must already exist before you can enter it in this property. The completion script runs after the verification script.

An automated activity also has a Transition Order window, where you can rearrange the possible transitions when there is more than one. You display the Transition Order window by clicking the Transitions tab in the Inspector window.

Transitions out of automated activities always default to true, because a true outcome means the process instance moves on to the transition. When you have more than one transition out of an automated activity, the activity operates in the same way as a decision point, which evaluates multiple conditions (for example, if the price is over $100 or less than or equal to $100) in a specific order. You set the order in the Transition Order window, by clicking the up and down arrows. The top condition in the window is evaluated first. Process Manager uses the first transition condition that is true to route the process instance.

Exception Manager If an error occurs, the Exception Manager invokes a new work item for correcting the error. Errors can occur in two cases: if a verification or completion script returns false, or if a runtime error occurs in a script (for example, in an assignment, expiration setter, or expiration handler script). See "Exception Manager" on page 116.

Subprocesses

A subprocess is a fully functional process that is called from within another process. The process that calls the subprocess is considered to be the parent process and the subprocess is considered to be its child process. A parent process can have several children processes, each of which is a stand-alone process complete with entry and exit points.

Subprocesses allow process designers to reuse processes across their environment. For example, if a bank has many different loan management processes, all of which require a credit check at a certain point in the process, each of the loan management processes can use the same credit check subprocess. Chapter 16, "The Loan Management and Credit History Applications" describes the Loan Management sample application and its subprocess the Credit History application; the two sample applications show how to use a subprocess in your applications.

Another advantage of subprocesses is that the logic flow of the main process map may be simpler to follow because a certain portion of its processing is hidden behind the subprocess icon. For example, in the Loan Management sample application, the Check Credit History processing is handled in the Credit History subprocess and is not exposed in the main Loan Management process map.

You can use a subprocess in the same way as an automated activity (see "Automated Activities" on page 105). You can place it anywhere on the process map, have multiple transitions coming out of it, and place complex scripts within the subprocess to perform actions that are hidden from the main process.

A timer agent coordinates how the parent and child process interact together. The parent process initiates a child process; when the child completes, it sets a value, wf_observer_url, in a Process database table. This is the URL of the parent process. A timer agent periodically checks this database table (by default, every five minutes). If there's a value in the wf_observer_url field, the timer agent attempts to connect the child to the parent process. Once the child successfully connects back to the parent, the value in the database table is set to null so the timer agent does not attempt to reconnect again.

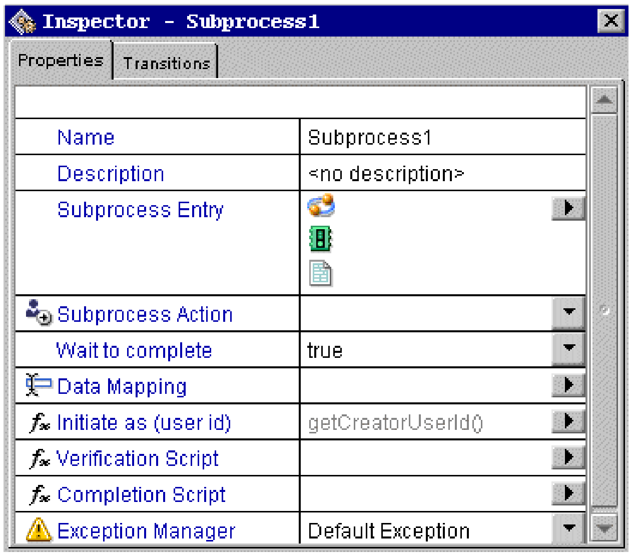
If there's a problem in connecting to the child or to the parent, an exception is thrown and the exception manager allows the administrator to intervene. See "Exception Manager" on page 116 for more information.

➤ **To set up a subprocess**

1. Create a child process.
2. Open the parent process map.
3. Drag a subprocess icon onto the parent process map.
4. Right-click the subprocess icon and choose Properties.

   The Subprocess Inspector Window appears, as shown in Figure 5-7.

Figure 5-7 The Inspector Window for a subprocess



Inspector - Subprocess1

Name	Subprocess1
Description	<no description>
Subprocess Entry	
Subprocess Action	
Wait to complete	true
Data Mapping	
Initiate as (user id)	getCreatorUserId()
Verification Script	
Completion Script	
Exception Manager	Default Exception

5. Fill in the properties for the subprocess.
6. Close the window.

   Your changes are saved automatically.

All subprocesses have the following properties:

Name The name of the subprocess.

Description An optional longer description of the subprocess.

Subprocess Entry The entry point in which the parent process enters the child process. To enter the subprocess entry:

1. Click the right arrow next to the subprocess icon. The Subprocess Entry Chooser window appears.
2. In the list of applications, click the plus (+) sign of the application you want to make a subprocess.
3. Choose an action or an entry point and click OK.

Subprocess Action The action that you follow from the Subprocess Entry field.

Data Mapping The data that must be mapped between the child and parent process so that values can be passed easily between them. To define the data mapping:

1. Click the right arrow in the Data Mapping field. The Data Mapping Setting window appears.
2. In the Mapping Script column, click the arrow for each row. The Field Mapping window appears.
3. Enter a static value, select a corresponding field from the parent data dictionary, or define a script and click OK.
4. When you have mapped all the data fields, click OK.

Initiator User Id The initiator of the subprocess.

You can use a dynamically derived value as the initiator. For example, in a hiring application that uses a child process to set up a secure ID card for each new hire, you identify a different new employee in each instance of the parent process and you want to use the new user IDs in the child process. In this case, you use a script to pull the new user ID from the corporate directory.

You can use the App User Id of the parent process as the creator of the child process. In this case, you leave the Initiator User Id blank because the default is to use the same user as the creator of the main process.

You can also use a static value as the initiator, which is especially useful when testing. The sample application, for example, uses a static value ("admin") to simplify setting up and deploying the application.

Verification Script The script that runs when the child process is completed. The script must already exist before you can enter it in this property.

Completion Script The script that runs when the child process is completed. The script must already exist before you can enter it in this property. The completion script runs after the verification script.

Exception Manager If an error occurs, the Exception Manager invokes a new work item for correcting the error. Errors can occur in two cases: if a verification or completion script returns false, or if a runtime error occurs in a script (for example, in an assignment, expiration setter, or expiration handler script). See "Exception Manager" on page 116.

Connecting the Parent and Child Process

When a parent process arrives at a child activity, it tries to connect to the subprocess and launch it. The parent process normally invokes the subprocess with its App User ID as the initiator of the subprocess. It can also invoke the subprocess with a different user id, but in this case, this user id needs to be in the trusted user group of the subprocess. This requirement is in place to avoid spoofing. To add an initiator id to the trusted user group of the subprocess, see “Creating Groups and Roles” on page 132.

Custom Activities

Custom activities are Java classes created by a developer that you can use in Process Builder applications to perform complex automated tasks, such as connecting Process Manager applications to external applications and databases.

Custom activities are similar to automated activities, in that they perform tasks without user intervention. However, custom activities, because they are Java classes, can be used for more complicated tasks than the JavaScript scripts of automated activities.

This section presents an overview of custom activities. For a detailed example, see the *Process Manager Programmer’s Guide*.

Using a Custom Activity

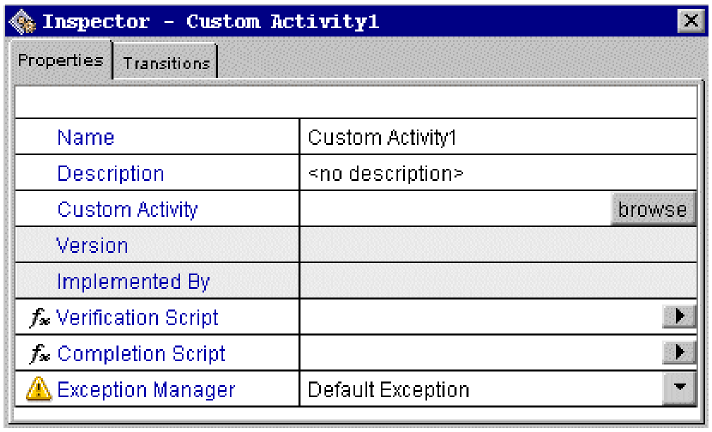
After you add a custom activity to the palette, you can use the custom activity in your process map. To do so, perform the following steps:

1. Drag the custom activity icon from the palette to the process map.
2. Right-click the custom activity icon on the process map and choose Properties. The Inspector Window appears.
3. Click the Input and Output tabs to make sure the data is mapped correctly between the custom activity and the application’s fields.

Custom Activity Inspector Window

Figure 5-8 shows the Inspector window for a custom activity when you first drag the activity to the process map.

Figure 5-8 Custom Activity Inspector



Name	Custom Activity1
Description	<no description>
Custom Activity	
Version	
Implemented By	
Verification Script	
Completion Script	
Exception Manager	Default Exception

Note that there are only two tabs in the window: Properties and Transitions. The Properties tab displays the following properties:

Name The name of the custom activity that appears in Process Builder, and in Process Express. Since this name appears on users' work lists, make it descriptive of the action they need to take.

Description An optional longer description of the activity that appears in Process Express.

Custom Activity The file that contains the Java class and its xml descriptor. Use Browse to go to an .xml, .zip, or .jar file to use. After you set this property, the Inspector window changes to reflect information defined by the custom activity you selected.

Version The custom activity's version number.

Implemented By The Java class that implements this custom activity.

Verification Script A script that runs when the step is completed. The script must already exist before you supply it as a property. The verification script runs immediately after the custom activity finishes executing.

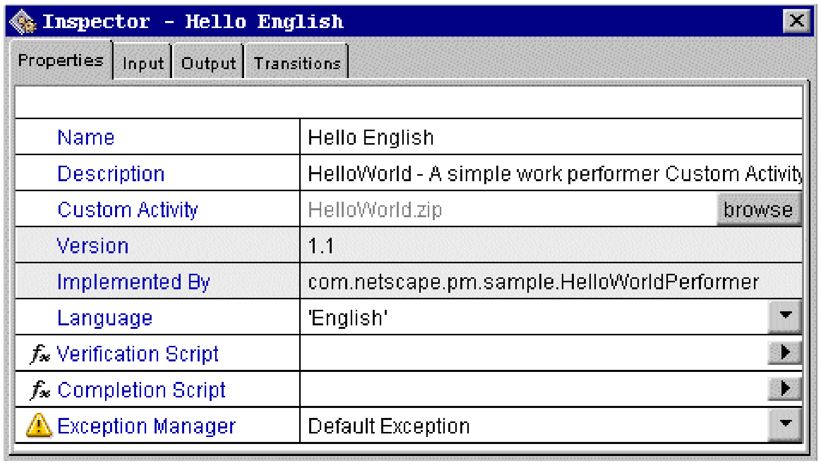
Completion Script A script that runs when the step is completed. The script must already exist before you supply it as a property. The completion script runs immediately after the verification script.

Exception Manager If an error occurs, the Exception Manager invokes a new work item for correcting the error. Errors can occur in two cases: if a verification or completion script returns false, or if a runtime error occurs in a script (for example, in an assignment, expiration setter, or expiration handler script). See "Exception Manager" on page 116.

Inspector Window After Setting a Custom Activity

Once you set the Custom Activity field, the Inspector window changes. An example is shown in Figure 5-9:

Figure 5-9 Custom Activity Inspector, after setting the Custom Activity property



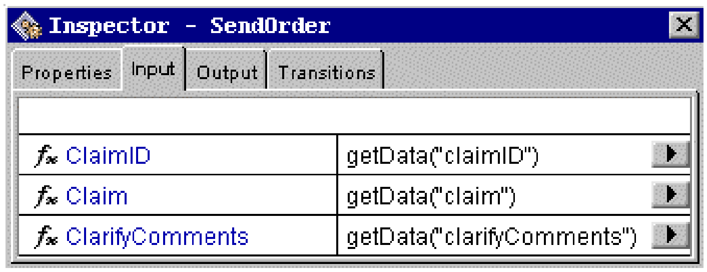
Name	Hello English
Description	HelloWorld - A simple work performer Custom Activity
Custom Activity	HelloWorld.zip
Version	1.1
Implemented By	com.netscape.pm.sample.HelloWorldPerformer
Language	'English'
Verification Script	
Completion Script	
Exception Manager	Default Exception

After setting the custom activity, if an environment parameter is set in the custom activity file, the parameter appears in the properties window. In Figure 5-9, Language is an environment parameter.

Two new tabs also appear: Input and Output.

The Input tab shows the parameter names in the custom activity's input hashtable, and shows how the parameter value is derived. Figure 5-10 shows an example of the Input tab.

Figure 5-10 The Input tab

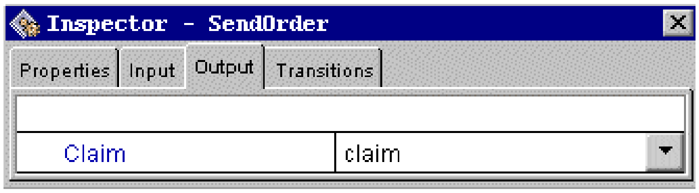


ClaimID	getData("claimID")
Claim	getData("claim")
ClarifyComments	getData("clarifyComments")

For example, the value for the input parameter `claimId` comes from getting the value in the application's `claimid` data field.

The Output tab shows the mapping between the output parameters in your custom activity and the data field in your application. Figure 5-11 shows an example of the Output tab.

Figure 5-11 The Output tab



Claim	claim

In this example, the value from the output parameter, `claimId`, is put into the application's data field `claimid`.

Adding a Custom Palette

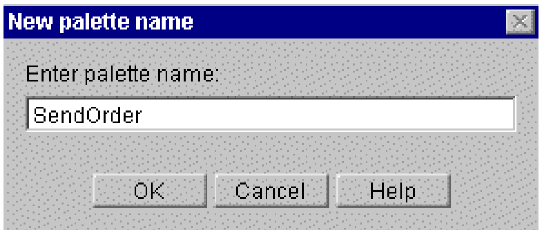
If you frequently use custom activities, you can add them to your palette, thereby giving you easy access to ready-made components.

➤ **To add a custom palette**

1. Right click on the Map Palette window and choose "Add custom palette."

   A dialog box appears, as shown in Figure 5-12:

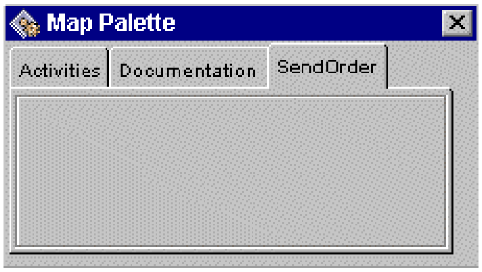
Figure 5-12 New Palette Dialog Box



2. Enter the name of the palette and click OK

   A tab containing the custom palette is added to the Map Palette, as shown in Figure 5-13.

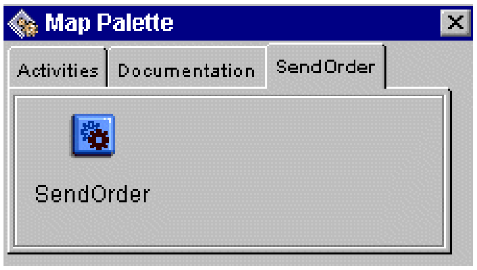
Figure 5-13 Blank custom palette



3. Add an item to the custom palette by right-clicking on the custom palette an choosing "Add custom activity."
4. Browse to the .xml, .jar, or .zip file that contains your custom activity.

   An icon representing the custom activity appears on the custom palette, as shown in Figure 5-14.

Figure 5-14 Custom palette with custom activity



5. To add the custom activity to your application, drag the icon from the custom palette to the process map.

For information about writing the XML and Java code for a custom activity, see the *Process Manager Programmer's Guide*.

Exception Manager

Exception managers allow users to intervene if errors occur in the application. You create an exception manager by dragging the exception manager icon to the process map. Most process map items represent steps in the process and are connected by transitions. Exception managers, however, are usually placed off to the side on process maps. They are not connected to other steps; instead, they are called by activities when an error occurs. You set up which exception manager is used at each activity using the Exception Manager property in the activity.

If a problem occurs while the work item is being processed, the exception manager assigned to that activity is called. The exception manager then generates a work item based on the exception manager properties. The work item is assigned to a user by the assignment script. Often the person assigned the work item is an administrator or the creator of the process instance, because these people are in a good position to diagnose what went wrong with the process.

You can attach a notification to the exception manager, which sends mail to the administrator when an exception is thrown. See "Notifications" on page 122 for more information.

Default Exception Manager

Every new process automatically has a default exception manager that appears in the application as soon as you create an activity that requires an exception manager. When you create new process map items that have Exception Manager properties, the properties default to the default exception manager.

To see the properties of the default exception manager, right-click the default exception manager in the application tree view and choose Properties. The default exception assigns the exception work item to the creator of the process instance. You can change the properties of the default exception handler, or you can create your own exception handler.

Creating an Exception Manager

➤ **To add your own an exception manager**

1. Drag an exception manager icon from the map palette onto the process map.
2. Right-click the exception manager icon and choose Properties. Update any properties as needed.
3. Assign this exception manager to a user activity, custom activity, automated activity, subprocess activity or end point.

   Right-click the activity icon and choose Properties to go to the Properties window. In the Exception Manager field, choose the exception manager from the drop-down list.
4. Either create a new form for the exception manager, or use an existing form. Because exception handlers don't have transitions, the button name is assigned automatically: Retry.

   Retry resubmits the work item where the error occurred.
5. Assign the form to the exception manager using the Form Access window.

Exception Manager Properties

All exception managers have the following properties:

Name The name of the exception manager.

Description An optional longer description of the exception manager.

Allow to Save True/false. When set to true, this option allows users to save a work item so that they can complete it at a later time. If you set this property to true, a Save button appears on the work item's in Process Express. Defaults to false.

Allow to Add Comment True/false. When you set this option to true, end users are allowed to add comments to the activity. The comment field appears automatically on forms presented to users for that activity in Process Express. The assignee can add comments, which are then displayed in Process Express as part of the process instance's history. Defaults to true.

Allow to Delegate True/false. When set to true this option allows users to delegate a step so that someone else can handle it for them. If a step is delegated, the person it is delegated to becomes the step assignee, with the associated access to forms. If you set this property to true, a "Delegate" button appears on the work item's form in Process Express. Defaults to false.

Assignment Script The script that assigns a user or group of users to a work item. When a user is assigned a work item, it appears in the user's work list in Process Express. You can use a predefined assignment script or create your own. If you create your own, the script must already exist before you can enter it in this property. See the programming information in this book for more information on creating scripts.

Expiration Setter Script The script that sets an expiration date or time for a work item. See "Setting Activity Expirations" on page 100 for further details about the script properties. The script must already exist before you can enter it in this property.

Expiration Handler Script The script that runs when a work item has not been completed by its expiration date or time. The script must already exist before you can enter it in this property.

Verification Script A script that runs when the step is completed. The script must already exist before you can enter it in this property.

Completion Script A script that runs when the step is completed. The completion script must already exist before you can enter it in this property. The completion script runs after the verification script.

Decision Points

Decision points are places where the process branches into different steps depending upon conditions. For example, if a price is above a certain amount, the process might branch to include an approval step that is unnecessary if the price is below a that amount.

Decision points have the following properties:

Name The name of the decision point that appears in Process Builder.

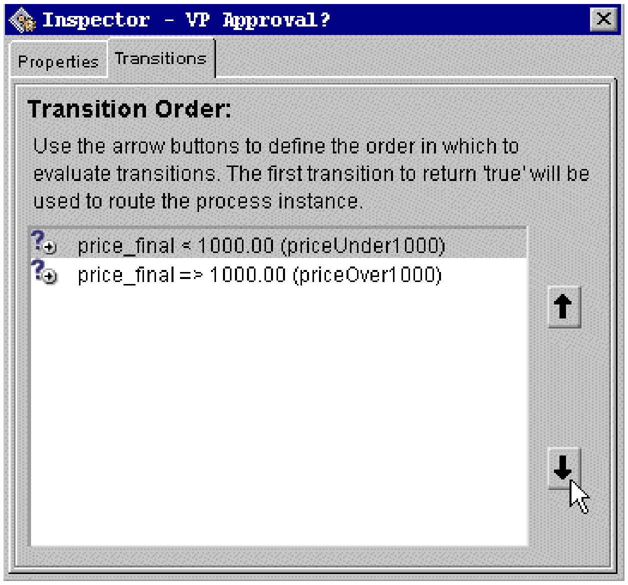
Description An optional longer description of the decision point.

Completion Script A script that runs when the step is completed. The script must already exist before you can enter it in this property.

You add possible outcomes to the decision point by adding transitions with conditions. For example, you could create two transitions, one if the price stored in the `price_final` field is over $1000, and another if the price is less than or equal to $1000. Make sure that all possible outcomes are covered in your transitions. For example, if you made transitions with the conditions that the price had to be greater than $1000 or less than $1000, you have not covered the condition of the price being equal to $1000. If you do not have options that cover all cases, the transaction rolls back and the user receives an error message.

Decision points also have a Transition Order window, where you can rearrange the transitions leading from the decision point. You display the Transition Order window by clicking the Transitions tab at the top of the Inspector window. An example from the DataSheet sample application is shown in Figure 5-15:

Figure 5-15 The decision point Transition Order window



Transition Order:

Use the arrow buttons to define the order in which to evaluate transitions. The first transition to return 'true' will be used to route the process instance.

price_final < 1000.00 (priceUnder1000)
price_final => 1000.00 (priceOver1000)

The top item in the window is evaluated first, but you can rearrange items by using the arrow icons. Process Manager uses the first transition condition that is true to route the process instance. So for the example above, the process checks first to see if the `price_final` field is less than 1000.00. If that is true, the process continues with that transition. If the `price_final` field is more than 1000.00, the first condition is false, so the second condition is now evaluated. If the second condition is true, the process continues with that transition.

For more information on transitions, see "Transitions" on page 124.

Split-Join (Parallel Processing)

Parallel processing allows a process to branch in two or more directions so that two or more activities can execute in parallel. In complex processes, you can nest parallel processes within a larger parallel process. Note that the activities in the nested process are considered to be part of the nested process, not the larger process; that is, if you nest process "a" within process "A", each with its own pair of split-join icons, the activities in the nested process must progress from split "a" to join "a" not to join "A."

Chapter 15, "The Office Setup Application" describes the Office Setup sample application, which controls the process of setting up an office for a new employee. In this sample application, each subtask is grouped into a processing branch that progresses independently of the other subtasks. For example, the MIS department can set up the phone while the purchasing department is ordering the computer. Problems completing one task won't affect the progress of a parallel task.

Properties of a Parallel Process

Parallel processes have the following properties:

Name The name of the split or join.

Description An optional longer description of the split or join.

Completion Script A script that runs when the step is completed. The script must already exist before you can enter it in this property.

Parallel processing also has evaluation order windows, where you can rearrange the transitions leading from the split or join icons. You reach the evaluation order window by clicking the evaluation order tab at the top of the inspector window. The top item in the evaluation order window is evaluated first. Rearrange items using the arrow icons.

Adding a Parallel Process

➤ **To use parallel processing in your application**

1. Drag a split-join icon onto the process map.
2. Between the split and the join icons, add the desired activities.
3. Drag transitions from the split to the activities. The transitions are similar to the conditions out of decision points and default to a value of true. For more information, see "Decision Points" on page 118.
4. You can reorder the evaluation order of the conditions as you would for conditions out of a decision point. For more information, see "Decision Points" on page 118.
5. Drag transitions from each of the activities in all branches from a split so that they eventually end up in the corresponding join.

The rules for using parallel processing activities are as follows:

- A branch (or thread) of activities out of a split must eventually complete in its corresponding join. Activities that are part of a processing branch out of a given split must progress through that branch until the corresponding join.
    - They cannot exit out of the processing branch directly to an exit point.
    - They cannot transition out of the processing branch to an activity that is part of another processing branch.
    - They cannot transition out of the processing branch to an activity that is part of a subordinate nested parallel process, which would be therefore part of another processing branch.
    - If they are in a nested parallel process within a larger, higher-level parallel process, they cannot transition to an activity that is in the higher-level parallel process.
    - They cannot transition back into the split icon that started the processing branch they are a part of.
    - They cannot transition back to the activity that precedes the split icon.
- If you delete a split or a join, the corresponding half icon is also deleted (you must also delete any transitions in or out of a split or join at the same time).
- After you deploy an application to test or production, you cannot delete a split or a join.
- You cannot loop on a split or a join.

- You can have conditions set on transitions that come out of a join. However, if the join fails, then the transaction rolls back to the most recent activity in the process block. Similarly, if no condition out of a split evaluates to true, then the process rolls back to the most recent activity.
- By default, when you first create a new split or join, the other half icon is given the same name. You can change this later to display different names.

Notifications

Notifications are email messages sent when the process reaches an activity. One use of notifications is to notify the person who needs to perform the next step in the process. Another use is to give a status (for example, information about the document's progress within the process) to the person who created the document. You can attach notifications only to user activities and end points.

When the end user receives a notification, the title of the notification is the application's title field, followed by the priority field. For more information on these fields, see "The Application Properties Dialog Box" on page 87.

Notification Properties

Notifications have the following properties:

Name The name of the notification that appears in Process Builder.

Description An optional longer description of the notification. For example, it could describe the step in the application where the notification happens.

Email Address(es) The email address for the notification. This can be a static email address, but more commonly would be a script that assigns an email address based on the instance of the process. For example, a script like `emailOfCreator` sends a notification to the email address of the person who created the process instance. If you want to use a static email address, you must put quotation marks (" ") around it. Multiple addresses are separated by commas.

Content Type The format the email message's content is in. You can choose between `text/html` (messages sent in HTML format) and `text/plain` (messages sent in plain text format). If your mail system does not support HTML mail, choose `text/plain`, or your notifications may not work. In both cases, the character set is `us-ascii`.

Email Body The content of the email notification you want to send. If you enter static text, you must put quotation marks (" ") around it. You can also enter the name of a script that returns the email message string.

CC The email addresses for people to be copied on the email message. See the description above for the Email Address(es) notification property.

Subject The subject of the email message. If you enter static text, you must put quotation marks (" ") around it. You can also enter the name of a script that returns the email subject string.

Character set The character set for the email message.

Built-in Email Notification Scripts

As shown in Table 5-2, Process Builder provides built-in scripts that you can use for addressing notifications.

Table 5-2 Built-in Email Scripts

Script	Definition
`emailOfRole(roleName)`	Sends email to the user who is performing a field role for a process instance. `roleName` is the name of the field role.
`emailOfAssignees()`	Sends email to the user assigned to the process step.
`emailOfCreator()`	Sends email to the creator of the process instance.
`emailByDN(userDN)`	Sends email to the specified user DN (distinguished name).
`emailById(userID)`	Sends email to the specified user ID

Exit Points

Exit points are steps at which a user can exit a process. An application can have several exit points. For example, a vacation request process might end with the vacation being approved or denied.

Exit points have the following properties:

Name The name of the exit point that appears in Process Builder.

Description An optional longer description of the exit point.

Exception Manager If an error occurs, the Exception Manager invokes a new work item for correcting the error. Errors can occur in two cases: if a verification or completion script returns false, or if a runtime error occurs in a script (for example, in an assignment, expiration setter, or expiration handler script). See "Exception Manager" on page 116.

Transitions

After you place items from the Palette on your process map, you must connect them to show how data flows or how actions flow. You connect steps with transition lines that have directional arrows indicating their destination item.

NOTE You cannot use transitions with exception handling. For more information, see "Exception Manager" on page 116.

Types of Transitions

There are two basic types of transitions:

- regular transitions (which do not depend upon conditions and are represented on the process map as blue lines)
- conditional transitions (which depend upon conditions and are represented on the process map as green lines)

Regular transitions originate from activities and do not have a property where you can set conditions for their use. If you have multiple regular transitions from an activity, the process branches. The user chooses between transitions them by selecting a button on a form.

Conditional transitions originate from automated activities and decision points. They are not executed until a condition is met. If you have multiple conditional transitions, they are executed based on evaluation order and condition. Process Manager evaluates the conditions in the order specified and the first condition of a transition that it finds to be true is the transition it executes.

NOTE In addition to continuing on to the next step in a process, a transition can loop back to a previous step. To draw a looping transition on a process map, drag the arrow icon backwards to a previous step. For example, if at one step in a process a graphic artist produces a graphic, and the next step is manager review of the graphic, two transitions can lead from the manager's approval step. The first transition, if the manager approves the graphic, continues to the next step in the process. The second transition, if the manager does not approve the graphic, loops back to the previous step so that the graphic artist works on the graphic again.

Adding a Transition

➤ **To add a transition**

1. From the arrow just beyond the upper-right edge of an icon on the process map, drag the arrow to another icon. Do not release the mouse button until you've moved the arrow to the inside of the destination icon.

   An arrow connecting the two steps appears on the process map. The arrows are blue if they begin in an entry point or activity, green if they begin in an decision point or automated activity.

2. Give the transition a descriptive name.

   This name appears as a button on the form (unless the transition is from an automated activity or a decision point).

3. Define the transition's properties in the Inspector Window.

Transition Properties

All transitions have the following properties:

Name The transition's name is the name of the button on the form for the activity at from which the transition originates. The name typically describes something about the activity that is the destination of the transition. For example, a line that goes from "Create Graphics" to "Review Datasheet" might be called "Publish for Review" because when the graphics are done, the art department wants to publish the new datasheet for review.

Description An optional longer description of the transition.

Setting the Property for a Virtual Transition

Transitions that lead from a user activity also have the following property:

Virtual Action If set to true, no button associated with this transition will appear on the assignee's HTML form. A common use for a virtual transition is for activities that expire. If you set up an activity so that it expires after a certain amount of time, the expiration handler script might use this virtual transition to advance the process to another step. The advance occurs through JavaScript, without the involvement of the assignee. For more information on using JavaScript to move from one process step to another, see the `moveTo` function in Appendix A, "JavaScript API Reference."

Setting the Property for a Conditional Transition

Conditional transitions (from automated activities and decision points) also have the following property:

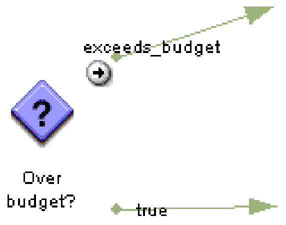
Condition The condition the activity must meet before executing the transition. Conditional transitions can be scripts that return true or false (you can type either the name of the script or type the script itself into the condition property). Conditions can also be based on values of data fields. For example, if your condition is that the price field be over 1000, then your condition can be `price>1000`, where `price` is the price field. If the value of the price field is over 1000, then this condition is true.

The condition defaults to true for automated activities and for custom activities. If there is more than one condition, you can prioritize the conditions in the evaluation order window. For more information on prioritizing conditions, see "Decision Points" on page 118.

Example Using a True/False Field

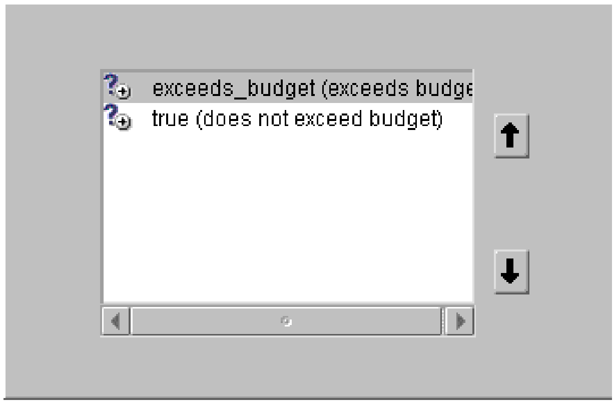
If you have a field with true and false as its valid values, you can use the field name as a transition's condition. An example is shown in Figure 5-16. Suppose you have a field named `exceeds_budget`, and the value can be true or false. In that case, the first condition is "`exceeds_budget`," and the second condition is "true."

Figure 5-16 Example of condition set to a field



To make sure these conditions are evaluated in the proper order, open the decision point's Inspector window and click the Transitions tab. This opens the Transition Order window, as shown in Figure 5-17.

Figure 5-17 Example of the Transition Order window for a decision point



exceeds_budget (exceeds budge
true (does not exceed budget)

The process evaluates the first condition, "exceeds_budget," to see if the value of the field is true or false. If the value of the field is true, then the condition is true, and the process continues with the "exceeds_budget" transition. If the value of the exceeds_budget field is false, the process evaluates the next condition. In this example, the next condition is true. Therefore, its outcome is always true, but it is evaluated last. A final condition of true works like an "else statement." That is, the true condition automatically advances the process if none of the previous conditions are true.

Chapter 6

Defining Groups and Roles

This chapter explains creating groups and roles and defining their properties.

This chapter includes these sections:

Groups and Roles Overview

For a Process Manager application, user roles and groups determine which users see which forms and which users perform which tasks. Scripts can also use the groups and roles. Groups are a collection of users in an application. Roles are the parts users play in a specific process instance. For example, you could have a group composed of all employees in the company. If someone in that group submits a time off request, for that particular process instance that employee has the role of creator. The creator role is defined automatically when you create a new application, however, you can create other roles. The user and group information in the corporate user directory is the source of user information when you set up groups and roles.

The following types of groups and roles are available:

Application Group A group defined in a particular application that points to user data in the corporate user directory. The group information is stored in the configuration directory, and the user information is stored in the corporate user directory. Because these groups are stored in the configuration directory, you do not need to have write access to the corporate user directory to create one. The context of this group is the application, since it cannot be used by other applications.

Corporate Group A group that points to a group defined for the corporation in the corporate user directory. The application's corporate group is exactly the same as the group in the corporate user directory; when the group in the corporate user directory is updated, the corporate group automatically uses those changes. Using corporate groups you can leverage the groups already defined in the corporate user directory. The context of this group is in the corporation; it can be used for many things besides applications.

Dynamic Group This group uses a filter to list members in the corporate user directory that match the attributes in the search filter. Unlike a corporate group, which is already defined as a group in the corporate user directory, the dynamic group uses corporate user directory information to create a group dynamically. This group is defined in the context of the corporation (it changes based on changes in the corporate user directory) but it also is used only for the application.

Field Role A role represented as a data field. The value of the field varies by process instance, so the role's context is the particular process instance. For example, the creator role (the person who starts a particular process instance) is a field role. The field role can only be one user; you cannot use a field role to assign an activity to a group.

Internal Group/Role An internal group or role is defined by default in Process Manager. For example, the all group, which represents all users in the corporate user directory, is an internal group.

There are three high-level steps in creating groups and roles:

1. Choose the type of group or role you want.

   You can do this by asking yourself what the context of the group or role is. Is it just for this application, or is it a group that has corporation-wide implications?

2. Choose valid users for the group or role.

3. Prioritize all of the groups and roles in an application.

You need to prioritize the groups and roles so that if a user is assigned to multiple roles, the form access system can determine what precedence to give each role. For more information, see "Prioritizing Groups and Roles" on page 145.

Default Groups and Roles

When you create an application, the following groups and roles are created by default:

all The group of all users in the corporate user directory.

creator The person who initiates a particular process instance.

admin The group of people allowed to administer the application's process instances using Process Business Manager. You need to add the names of the people who can administer the application to the application's default "admin" group. These people must also belong to the WF Admin Auth ACL for the Enterprise Servers. Anyone who is allowed to administer by the WF Admin Auth ACL can administer the application at the application level, but only users who belong to the application's "admin" group can administer the application at the process instance level.

trusted users A group that can start a subprocess as another user. The parent process redefines the initiator of the child process. If no one will need to start a subprocess as someone else, you need not assign anyone to the trusted users group.

You can customize some aspects of these groups and roles, but you cannot delete them. If you don't need them for your particular application (for example, if your application deals with sensitive information that you do not want to expose to the "all" group) you don't need to define forms for particular groups or roles. See "Setting Access to Forms" on page 182 for more information.

In addition to the above groups and roles, each application has an assignee role. It is set for every step in the process in which someone must perform an action. However, since it is a special role, it doesn't appear in the application tree view and you cannot set properties for it. It works in concert with other groups and roles. For example, a step may be assigned to a group, but the actual user from that group who performs the task in a particular process instance is the assignee.

Creating Groups and Roles

➤ **To begin defining a role or group**

1. From the Insert menu, choose Group & Role.
2. In the Create a New Role or Group dialog box, choose the type of role and fill in the fields.

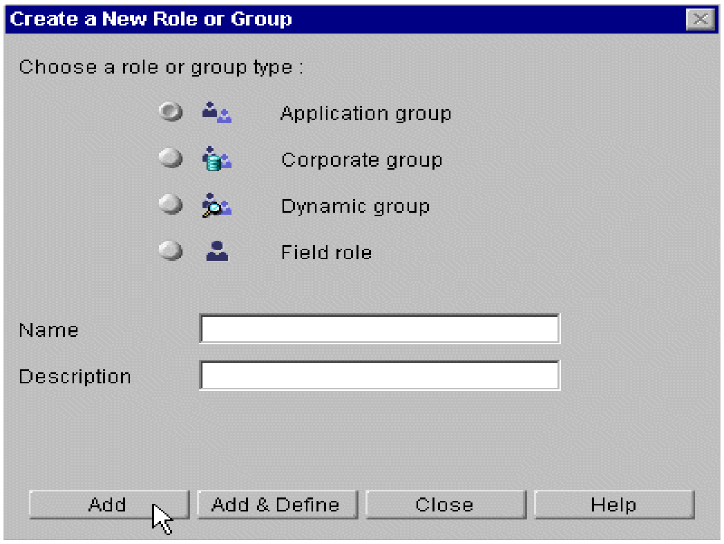
   See "The Create a New Role or Group Dialog Box" on page 132 for details.
3. Click either the Add button or the "Add & Define" button.
   - Click Add & Define if you want to define the new item right away. A dialog box for the type of group or role you chose (for example, an application group) appears.
   - Click Add to define the item later. The Create a New Role or Group dialog box remains open, and you can add additional items.
4. If you clicked Add & Define, fill in the fields of the next dialog box that appears.

   When you close the dialog box, your changes are saved automatically.

The Create a New Role or Group Dialog Box

To supply basic information about the role or group you want to create, enter this information in the "Create a New Role or Group" dialog box, as shown in Figure 6-1.

Figure 6-1 The Create a New Role or Group dialog box



Types of Groups and Roles

There are four different types of groups and roles you can create:

Application group A group defined in a particular application that points to user data in the corporate user directory. The group information is stored in the configuration directory, and the user information is stored in the corporate user directory. Because these groups are stored in the configuration directory, you do not need to have write access to the corporate user directory to create one. The context of this group is the application. Therefore, this group cannot be used by other applications.

Corporate group A group that points to a group defined for the corporation in the corporate user directory. The application's corporate group is exactly the same as the group in the corporate user directory; when the group in the corporate user directory is updated, the corporate group automatically uses those changes. Using corporate groups you can leverage the groups already defined in the corporate user directory. The context of this group is in the corporation; it can be used for many things besides applications.

Dynamic group This group uses a filter to list members in the corporate user directory that match the attributes in the search filter. Unlike a corporate group, which is already defined as a group in the corporate user directory, the dynamic group uses corporate user directory information to create a group dynamically. This group is defined in the context of the corporation (it changes based on changes in the corporate user directory) but it also is used only for the application.

Field role A role represented as a data field. The value of the field varies by process instance, so the role's context is the particular process instance. For example, the creator role (the person who starts a particular process instance) is a field role. The field role can only be one user; you cannot use a field role to assign an activity to a group. Because it's related to a datafield, you cannot create a Field Role if your application has been deployed for testing or production.

Name Enter the name of the role or group. This field cannot contain the following characters: quotation marks ("), commas (,), plus signs (+), and semicolons (;). If you are creating a field role, the name cannot be more than 18 characters long, because the name is also used as the name of the field.

Description An optional longer description of the role or group.

Adding a New Group or Role

To add a new group or role, click either the Add button or the "Add & Define" button.

Use the Add button to add several items in succession, without defining their properties. Each item will be added to the application tree view, but the "Create a New Role or Group" dialog box will remain open so that you can add subsequent items. After adding the desired items, you can define them by double-clicking them in the application tree view.

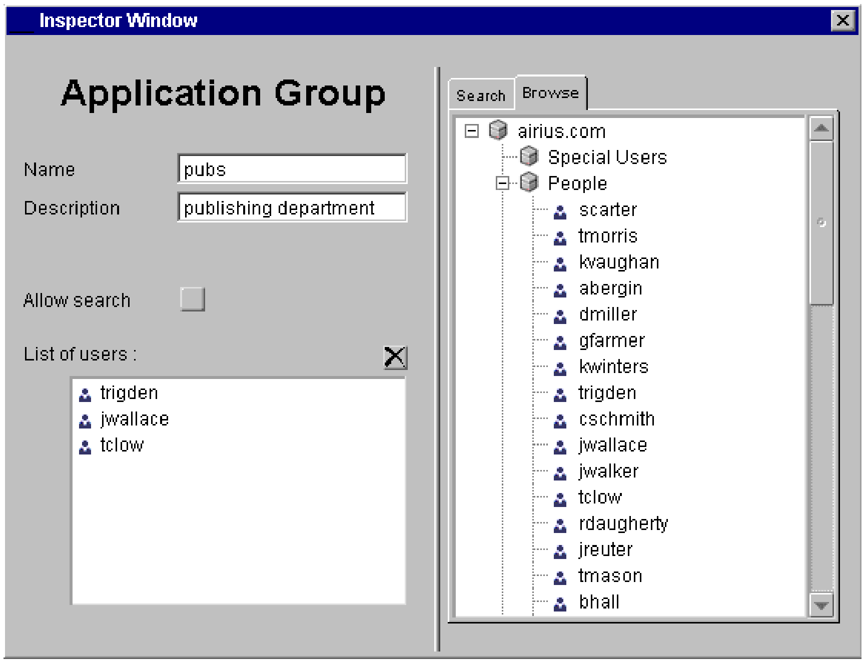
Use the "Add & Define" button to add an item and then immediately define it. Each item is added to the application tree view, but then a new dialog box appears, depending on the kind of group or role you are adding. One of the following dialog boxes will appear:

- "The Application Group Dialog Box"
- "The Corporate Group Dialog Box"
- "The Dynamic Group Dialog Box"
- "The Field Role Dialog Box"

The Application Group Dialog Box

The Application Group dialog box, shown in Figure 6-2, is where you identify which users are assigned to an application group. For example, if you are designing a time-off request application and have a role for HR approval, you could create an application group with the names of everyone in HR who is allowed to approve a time-off request.

Figure 6-2 The Application Group dialog box with Browse tab


Inspector Window

Application Group

Name: pubs

Description: publishing department

Allow search: ☐

List of users :
- trigden
- jwallace
- tclow

The dialog box contains the following fields, which contain the values you entered in the Create a New Role or Group dialog box:

Name The name of the group.

Description The group's description.

Allow search If checked, allows users in this group to search. If you do not make the group searchable, members of the group will not be able to use the search functionality to check the status of process instances or to find process instances related to specific criteria. They can still use the global search, though. See Chapter 13, "Setting Up Searching" for more information.

List of users The list of people that are part of this group. This list displays the user RDNs (relative distinguished name) as stored in the corporate user directory. You can add users to the List of Users from the corporate user directory list in the right pane. You can use the Search tab in the right pane to find users, or user the Browse tab to find them.

NOTE If no users are appearing in your Browse tab or when you search, make sure you have set a corporate user directory in the application's properties.

To see users in the Browse tab, expand the directory and groups by clicking the expansion icons (plus signs). The first time you click the icons the data loads from the corporate user directory. After that, a single click reloads cached data. To reload the information from the corporate user directory, double-click the directory and groups.

If you use the Browse tab, drag the users to the List of Users.

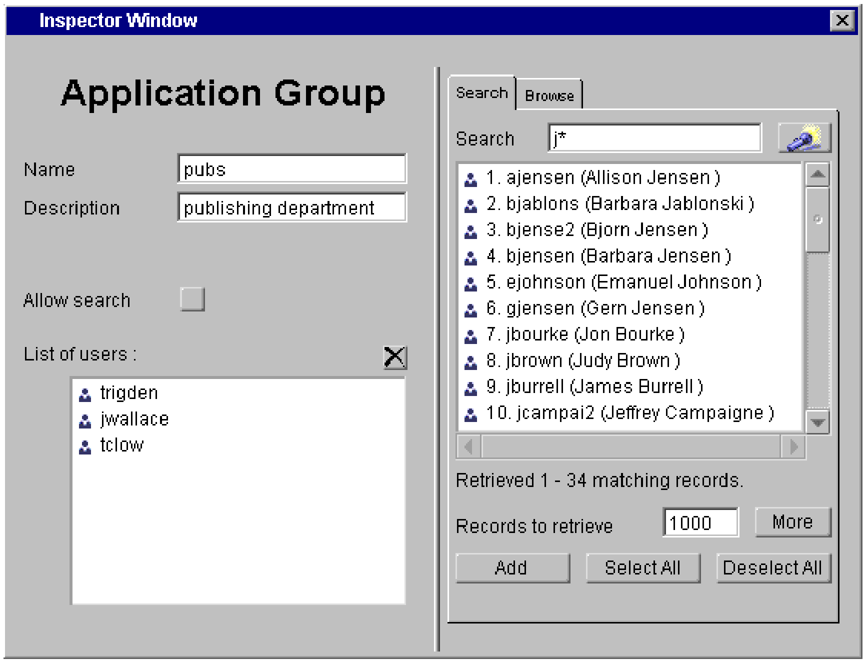
If you use the Search tab, use the buttons at the bottom of the right pane to add users to the List of Users. There are also Select All and Deselect All buttons.

To delete a user from the list of users, select the user name in the List of Users area and click the "X" box above the list.

➤ **To use the Search tab to quickly find and add users based on a wildcard pattern**

1. Click the Search tab, which is shown in Figure 6-3.

Figure 6-3 The Application Group dialog box with Search tab



Inspector Window

Application Group

Name: pubs

Description: publishing department

Allow search

List of users :

- trigden
- jwallace
- tclow

Search | Browse

Search: j*

1. ajensen (Allison Jensen)
2. bjablons (Barbara Jablonski)
3. bjense2 (Bjorn Jensen)
4. bjensen (Barbara Jensen)
5. ejohnson (Emanuel Johnson)
6. gjensen (Gern Jensen)
7. jbourke (Jon Bourke)
8. jbrown (Judy Brown)
9. jburrell (James Burrell)
10. jcampai2 (Jeffrey Campaigne)

Retrieved 1 - 34 matching records.

Records to retrieve: 1000

2. Enter a search pattern (such as j* or *jan*).

This pattern can be for any part of the user's name, such as first name, last name, or user ID.

3. Click the Search icon to get a list of all users that match the pattern.

By default, the list displayed is sorted by user RDN (relative distinguished name). You can specify the sort order in the `preferences.ini` file, which is located in the `builder` folder. Add the following line to the file:

`sortAttribute =` *attribute*

where *attribute* is any LDAP attribute, such as `cn` or `uid`, that you want to appear first in each line of the list. For instance, if you entered `sortAttribute = cn`, the common name appears first in each line displayed.

The search returns the number of records specified in the "Records to retrieve" box. By default, the amount of records retrieved is 1000. You can modify this default in the `preferences.ini` file; add the following line to the file:

`defaultSearchSize =` *number*

where *number* is the amount of records you want displayed by default.

4. Select one or more users from the list.

   To select multiple users, highlight a user and without releasing the mouse button brag to the last user you want to add, then release.

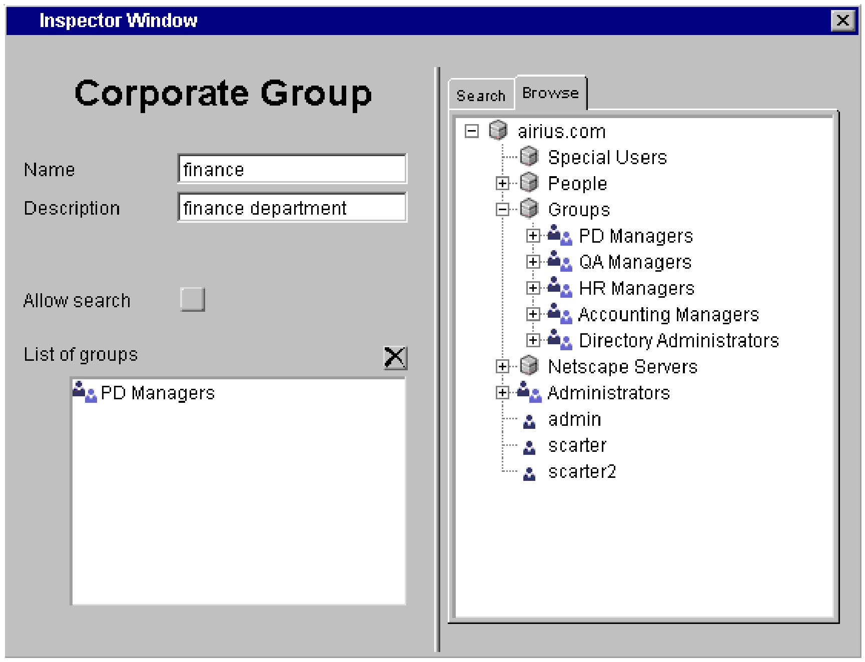
5. Click Add.

The Corporate Group Dialog Box

The Corporate Group dialog box, shown in Figure 6-4, is where you identify a corporate user directory group or set of groups as a group in your application. For example, if you want to have a group called "HR" that consists of all employees in the HR department, and a group like that already exists in the corporate user directory, you can create your group quickly by using the corporate group. Because it is tied to the corporate group, any changes made to the group in the corporate user directory are reflected in the application. Also, since the corporate groups are defined outside of applications, you can use the same group easily for multiple applications.

However, you cannot add or delete users from a corporate group in your application. You must use the corporate group exactly as it is set up in the corporate user directory. To manage your corporate directory, use the administration features of your Directory Server.

Figure 6-4 The Corporate Group dialog box with Browse tab



Name: finance
Description: finance department
Allow search: ☐

List of groups:
- PD Managers

- airius.com
  - Special Users
  - People
  - Groups
    - PD Managers
    - QA Managers
    - HR Managers
    - Accounting Managers
    - Directory Administrators
  - Netscape Servers
  - Administrators
  - admin
  - scarter
  - scarter2

The dialog box contains the following fields:

Name The name of the group.

Description The group's description.

Allow search If checked, allows users in this group to search. If you do not make the group searchable, members of the group will not be able to use the search functionality to check the status of process instances or to find process instances related to specific criteria. They can still use the global search, though. See Chapter 13, "Setting Up Searching" for more information.

List of groups The list of groups that are part of this group. You can add a group or groups from the corporate user directory list in the right pane to the List of Groups. You can use the Search tab in the right pane to find groups, or use the Browse tab to find them.

NOTE If no users are appearing in your Browse tab or when you search, make sure you have set a corporate user directory in the application's properties.

To see users in the Browse tab, expand the directory and groups by clicking the expansion icons (plus signs). The first time you click the icons the data loads from the corporate user directory. After that, a single click reloads cached data. To reload the information from the corporate user directory, double-click the directory and groups.

If you use the Browse tab, drag the groups to the List of Groups.

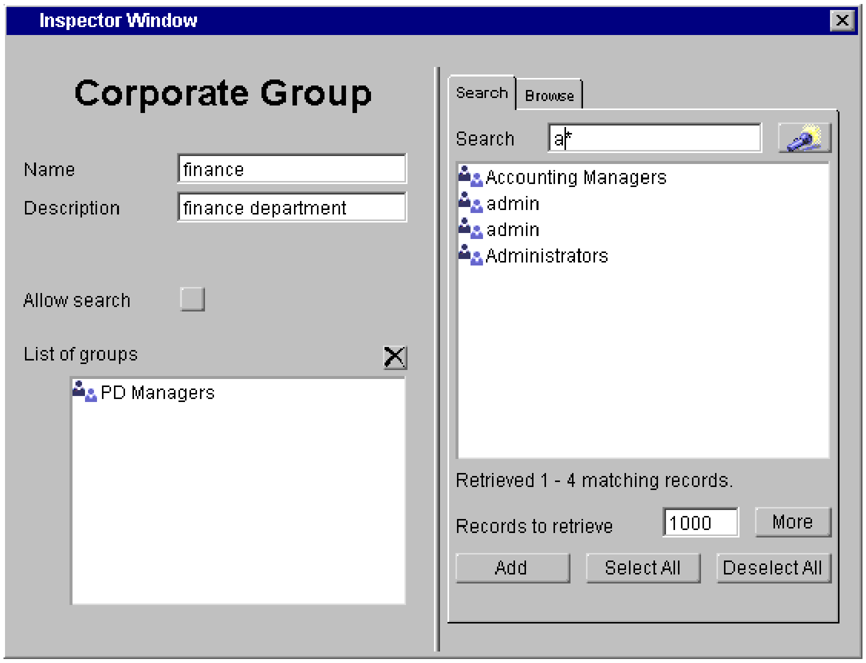
If you use the Search tab, use the buttons at the bottom of the right pane to add groups to the List of groups. There are also Select All and Deselect All buttons.

To delete a group from the List of groups, select the user name and click the "X" box above the List of groups area.

➤ **To use the Search tab to quickly find and add groups based on a wildcard pattern**

1. Click the Search tab, which is shown in Figure 6-5.

Figure 6-5 The Corporate Group dialog box with Search tab



Name: finance
Description: finance department
Allow search: ☐
List of groups:
PD Managers

Search: a*
Accounting Managers
admin
admin
Administrators

Retrieved 1 - 4 matching records.
Records to retrieve: 1000

2. Enter a search pattern (such as a* or *adm*).
3. Click the Search icon to get a list of all users that match the pattern.

   By default, the list displayed is sorted by user RDN (relative distinguished name). You can specify the sort order in the `preferences.ini` file, which is located in the `builder` folder. Add the following line to the file:

   `sortAttribute =` *attribute*

   where *attribute* is any LDAP attribute, such as `cn` or `uid`, that you want to appear first in each line of the list. For instance, if you entered `sortAttribute = cn`, the common name appears first in each line displayed.

   The search returns the number of records specified in the "Records to retrieve" box. By default, the amount of records retrieved is 1000. You can modify this default in the `preferences.ini` file; add the following line to the file:

   `defaultSearchSize =` *number*

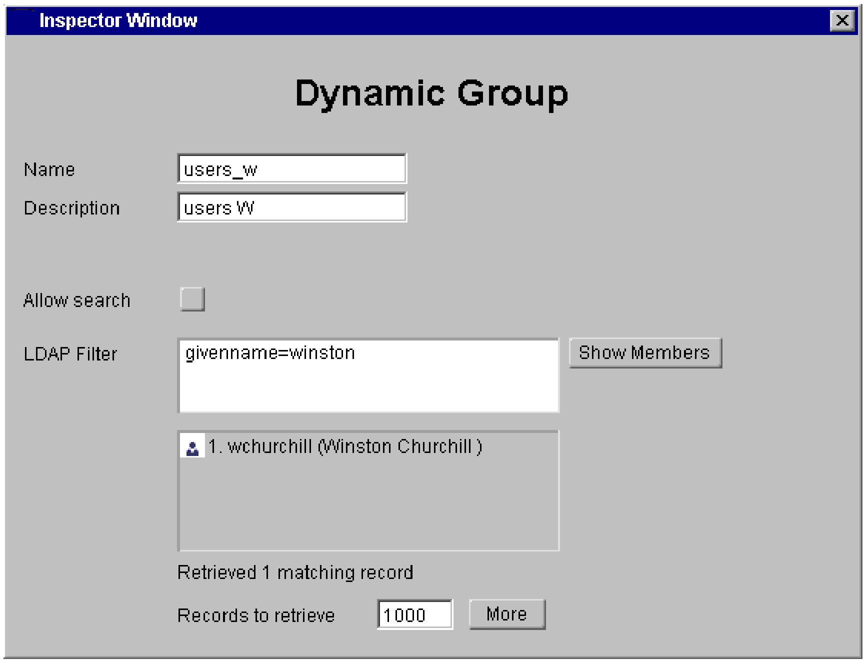
   where *number* is an integer number of records you want displayed by default.
4. Select one or more groups from the list.

   To select multiple users, highlight a user and without releasing the mouse button brag to the last user you want to add, then release.
5. Click Add.

The Dynamic Group Dialog Box

The Dynamic Group dialog box, shown in Figure 6-6, is where you define groups that are created dynamically. The application searches the corporate user directory using an LDAP filter, letting you take advantage of attributes in the corporate user directory.

Figure 6-6 The Dynamic Group dialog box



Dynamic Group

Name: users_w
Description: users W
Allow search: ☐
LDAP Filter: givenname=winston

1. wchurchill (Winston Churchill)

Retrieved 1 matching record

Records to retrieve: 1000

The dialog box contains the following fields:

Name The name of the group.

Description The group's description.

Allow search If checked, allows users in this group to search. If you do not make the group searchable, members of the group will not be able to use the search functionality to check the status of process instances or to find process instances related to specific criteria. They can still use the global search, though. See Chapter 13, "Setting Up Searching" for more information.

LDAP Filter Enter an LDAP filter and click Show Members to see a list of matches. You do not need to use the Show Members button, but it shows you what your filter fins. Process Builder does not check that the filter you enter is valid. If, after entering the filter string, nothing appears when you click Show Members, either your filter is invalid or it is a valid filter but there are no results that meet the criteria.

A search filter lets you search for an attribute in the corporate user directory. Here are a few sample searches you might use in this field:

- `manager=smith*`

  Searches for all users whose manager's user ID is smith* This syntax only works if the corporate user directory is set up to with the user ID as the RDN's first attribute. If the common name is first, the filter is manager=cn=smith*

- `DepartmentNumber=444`

  Searches for users whose department number is 444

- `(&(manager=smith*)(employeetype=employee*))`

  Searches for people whose manager's user ID is smith* and whose employee type is employee. The ampersand character (&) is the and operator.

- `(|(manager=smith*)(manager=jones*))`

  Searches for people whose managers are either smith* or jones*. The pipe (`|`) character is the or operator.

- `(&(givenname>=j*)(givenname<=o*))`

  Searches for people whose names begin with j, k, l, m, n, and o. The less than (<) and greater than or equal to (<=) characters specify the range.

For more information on LDAP filters, see the *iPlanet Directory Server Administrator's Guide*.

Records to retrieve The search returns the number of records specified in the "Records to retrieve" box. To obtain the next specified number of records, you click the More button. By default, the amount of records to retrieve is 1000. You can modify this default in the `preferences.ini` file, which is located in the `builder` folder. To modify this value, add this line to the file:

`defaultSearchSize =` *number*

where *number* is the amount of records you want displayed by default.

The Field Role Dialog Box

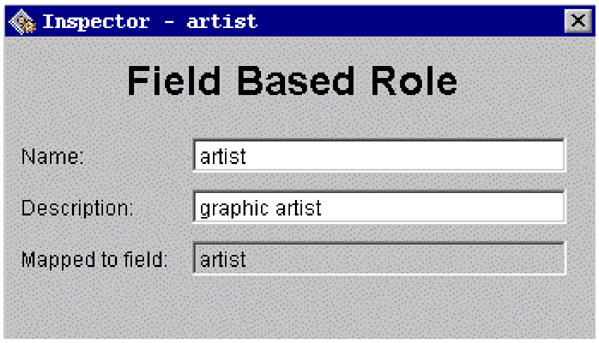
A field role is a role represented as a data field. For a particular process instance, the application uses the value in the data field to determine the user associated with the field role. When you create a field role, Process Builder automatically creates a data field to store the user role.

A field role can be used in an application in either of two ways:

- You can have the end user select a user for the role through the userpicker widget on a form.
- You can set the field role using the functions `setRoleById` or `setRoleByDN`.

Figure 6-7 shows the Field Role dialog box:

Figure 6-7 The Field Role dialog box



Name: artist
Description: graphic artist
Mapped to field: artist

The dialog box contains the following fields:

Name The name of the role. This name also becomes the name of the mapped field. The value of this property must be unique among the groups, roles, and data fields that are defined within a given application.

The name can contain only alphanumeric characters with no spaces. It cannot be longer than 18 characters. It's best to use all lower case, and use the underscore character (_) as a separator. Also, do not use a name that is a reserved SQL keyword (for example, select, integer, etc.), or the LiveWire reserved words project, request, server, and client.

Description The role's description.

Mapped to field The database field to which you are mapping this role. This field is created automatically, and has the same name as the role. It has the class ID TextField.

Prioritizing Groups and Roles

After setting up your groups and roles, you can prioritize them. Usually they are prioritized from most specific to most general. A user can belong to more than one group or role in the same application. Therefore, the priorities determine which forms the user gets.

The priority you set for Groups and Roles determines what order they appear in the Form Access window. See "Setting Access to Forms" on page 182 for more information. To set the order, perform the following steps:

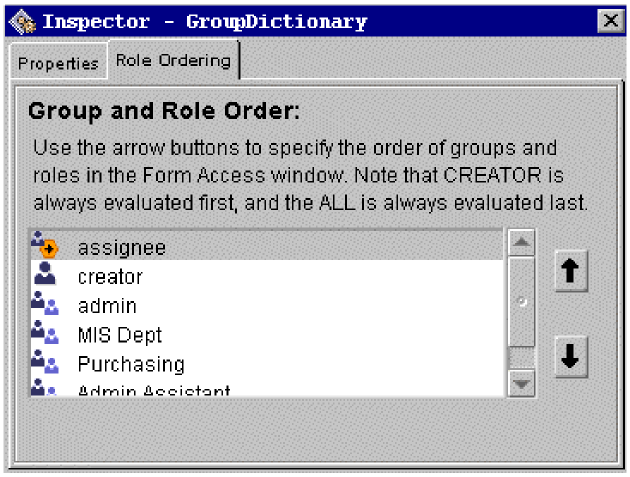
1. In the application tree view, double-click the Groups and Roles folder.

   The Inspector window appears.

2. In the Inspector window, click the Transitions tab.

   You see all the groups and roles listed, as shown in Figure 6-8:

Figure 6-8 Group and Role Order window



Group and Role Order:

Use the arrow buttons to specify the order of groups and roles in the Form Access window. Note that CREATOR is always evaluated first, and the ALL is always evaluated last.

assignee
creator
admin
MIS Dept
Purchasing
Admin Assistant

   The `assignee` role is always at the top of the list, and the `all` role is at the bottom. You cannot change the priorities for these two.

3. To change the priority of a group or role, select it.

4. Click the Up and Down arrows to move the groups and roles. Items at the top of the list have the highest priority.

   For example, a user who belongs to both the HR Dept group and the Manager group in the example above gets the form for the HR Dept if it exists. If not, then the user gets the form for the Manager group. If that form doesn't exist, then the user gets the form for "all."

5. Close the window when you're done. Your changes are saved automatically.

The order you set in this window shows up in the Form Access window for forms.

Deleting Groups and Roles

➤ **To delete a group or role**

1. In the application tree view, select the group or role.
2. Right-click and choose Delete, or choose Delete from the File menu.

After deleting a field role, you must also delete the corresponding field. However, you cannot delete a field if you have already deployed your application to the Production stage.

Chapter 7

Defining Data Fields

This chapter explains how to create data fields for an application and describes the classes of data fields that are available to you.

This chapter includes these sections:

Data Field Overview

The application's data dictionary contains all the data fields that are used by the application and stored in the database. Once you've defined your process map and groups and roles, you start defining the information that is captured and routed throughout the application. If you have a process that currently uses paper forms, a good start is to look at the information gathered on those forms.

A data field can be a structured field in a database, a document, or a file. The data dictionary contains all fields used by an application, regardless of which users see them at which steps in the process.

A field defines how the user interacts with the data (the class), and how the data is stored in the system (the type). Process Builder has some predefined field class IDs, but you can also create your own to extend the field framework.

NOTE If you are using subprocesses or custom activity you may need to map the data between the parent and child process. See "Subprocesses" on page 107 and "Custom Activities" on page 111 for more information.

Creating a Data Field

To add a field to the data dictionary, you should first ask yourself how the field interacts with the user (which determines the class) and what kind of information the field will store (which determines the type).

➤ **To define a field**

1. From the Insert menu, choose Data Field.
2. Fill out the fields in the Create a New Data Field dialog box.

   Each data field must have a unique name. The steps are described briefly below; see "The Create a New Data Field Dialog Box" on page 149 for more information.
3. Add a new data field using one of these options:
   - Add a custom data field from predefined classes. Click the radio button Custom Data Field and choose a Class ID from the drop-down list.
   - Add a custom data field from a class you create yourself. Click Add New Class, browse to the .zip or .jar file that contains the class you created, and click Open. The class appears in the drop-down list. For information on writing a custom field, see the *Process Manager Programmer's Guide*.
   - Add a predefined data field from a template. Click the radio button Predefined Data Field and choose a predefined type from the drop-down list.
4. Choose a class ID or template.

5. Click either the Add button or the "Add & Define" button.
   - Click Add & Define if you want to define the new item right away. The Inspector window appears for the data field you added.
   - Click Add to define the item later. The Create a New Data Field dialog box remains open, and you can add more items.
6. If you clicked Add & Define, define the properties in the Inspector window that appears.

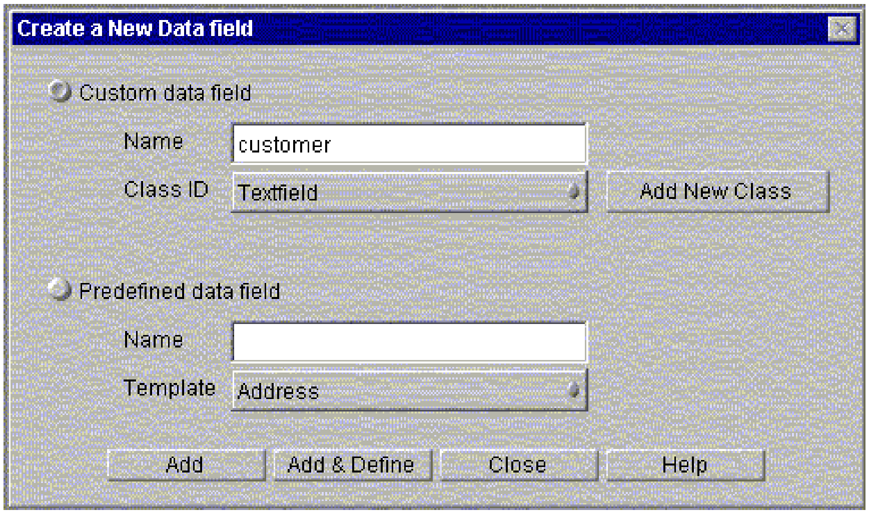
   When you close the window, your changes are saved automatically.

The new data field is now listed in the Data Dictionary folder in the application tree view.

The Create a New Data Field Dialog Box

The fields you fill out in this dialog box (see Figure 7-1) depend upon whether you are creating a custom data field or a predefined data field. After you add a field, you can define its properties in the Inspector window.

Figure 7-1 The Create a New Data Field dialog box


Create a New Data field

Custom data field
Name: customer
Class ID: Textfield

Predefined data field
Name:
Template: Address

Creating a Custom Data Field

To create a custom data field, enter values for the following fields:

Name of this Field The name of the data field. It is also used to name the database column that is going to store this field information. The value of this property should be unique among the data fields that are defined within a given application.

The name can contain only alphanumeric characters with no spaces. It cannot be longer than 18 characters. It's best to use all lower case, and use the underscore character (_) as a separator. Also, do not use a name that is a reserved SQL keyword (for example, select, integer, etc.), or the LiveWire reserved words project, request, server, and client, or the word comment.

See Appendix B, "Reserved Words" for a list of Process Manager reserved words that must not be used as data field names.

Class ID The class of the field you are creating. Technically, the class ID is the Java component that implements the field's format, but you don't need to understand the Java components unless you want to create your own Class ID. To add your own class IDs, click the Add New Class button and add your class to the drop-down list of class IDs. For information on writing a custom field, see the *Process Manager Programmer's Guide*.

Creating a Predefined Data Field

To create a predefined data field, enter values for the following fields:

Name of this Field defines the name of the data field. It is also used to name the database column that is going to store this field information. The value of this property should be unique among the data fields that are defined within a given application.

The name can contain only alphanumeric characters with no spaces. It cannot be longer than 18 characters. It's best to use all lower case, and use the underscore character (_) as a separator. Also, do not use a name that is a reserved SQL keyword (for example, select, integer, etc.).

Template A predefined field created with default data that you can modify. These templates make defining a field easier if you are inserting a field of a specific type: address, telephone number, or name.

Adding a Data Field

To add a new data field, click either the Add button or the “Add & Define” button.

Use the Add button to add several items in succession, without defining their properties. Each item will be added to the application tree view, but the “Create a New Data Field” dialog box will remain open so that you can add subsequent items. After adding the desired items, you can set their properties by double-clicking them in the application tree view.

Use the “Add & Define” button to add an item and then immediately define it. Each item is added to the application tree view, but the Inspector window also appears.

Setting Field Properties

Data fields are listed in the Data Dictionary folder in the Application Tree View window.

➤ **To set or modify field properties**

1. In the application tree view, in the Data Dictionary folder, double-click the field name, or highlight it and click the Inspector icon.
2. Update the properties.

   Some properties are grayed out and you cannot change them. Other properties contain values in angle brackets (<>). Those values are placeholder values. Replace the brackets and the values inside them with your own data.
3. Close the window. Your changes are saved automatically.

The following properties are available for all fields. In addition, fields have properties determined by their class ID. For more information about those additional properties, see the documentation for those properties.

Data Type (Required) Defines the data type of the field (how it is stored in the database). Valid values depend on the Field Class ID and include TEXT, LONGTEXT, INT, FLOAT, DATE, DATETIME, and FILE. These data types map directly to the SQL data types with the exception of FILE. For more information, see your database documentation.

Display Name The name used in Process Express when end users are allowed to search on a field. Use a name that will be clear and meaningful for the end user.

Field Class ID (read only) This property describes the format of the data field as it appears on the form.

Help Message A help message associated with a data field. When users click the field in a form, this message will appear as text at the bottom of the browser window (below the scroll bar).

Name (read only) The data field's name.

Short Description A description of the field. Currently, this property is unused.

Custom Data Fields with Predefined Class IDs

Use this option to design your own fields using the class IDs that are predefined in Process Builder:

- "CheckBox"
- "Computed"
- "Date"
- "DateTime"
- "Digital Signature"
- "File Attachment"
- "Password"
- "Radio Buttons"
- "Select List"
- "TextArea"
- "TextField"
- "URL"
- "UserPicker Widget"

The following sections describes these class IDs in more detail.

CheckBox

Use this class ID to create fields in the form of checkboxes. In addition to the properties listed in “Setting Field Properties” on page 151, CheckBox fields have the following properties:

Default Value The default value of the CheckBox field. Valid values are checked and unchecked.

Label The label that appears to the right of the checkbox. If you would like to create the label by typing it on the form in the form editor, leave this property blank. However, any label you type into this property is automatically included any time you put the field on a form. If you choose to type the label on the form, you need to type it on each form that uses this field.

On click A script that runs every time the user clicks on a checkbox. The script is client-side Javascript that is associated with the onClick event handler of the field.

Computed

This class ID is for fields whose values are computed by the application, not entered by the users. In addition, users cannot view these fields. If you want the field to be viewable, use a text field instead.

In addition to the properties listed in “Setting Field Properties” on page 151, Computed fields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false.

Default Value Defines the default value of the field.

Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. It must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

Date

This class ID lets the user enter and validate a date.

In addition to the properties listed in "Setting Field Properties" on page 151, Date fields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false

Date Format Represents the formatting of the date. Valid values are DD/MM/YYYY or MM/DD/YYYY.

Default Value Defines the default value of the field.

DateTime

This class ID lets the user enter and validate a date and time.

In addition to the properties listed in "Setting Field Properties" on page 151, DateTime fields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false

DateTime Format Represents the formatting of the date and time. Valid values are MM/DD/YYYY HH:MI:SS or YYYY/DD/MM HH:MI:SS. MM is the month, DD is the day of the month, YYYY is the year. HH is the hour, MI is the minutes, and SS is the seconds of the time. Note that the user must put a space between the date and the time.

Default Value Defines the default value of the field.

Digital Signature

The Digital Signature field is used for signing data. If the user has a certificate, the data is encrypted. For example, you would want a Digital Signature field if a Vice President has to approve a price. In this example, the price field is a user input field that is included in the List of Signed fields for the VP Signature field. When the VP updates the signature field, all the fields listed in the List of Signed fields are signed.

A Digital Signature field can be signed only by one person for a process instance. Once someone signs the field, the next time the field appears, it shows the name of the person who signed it.

In addition to the properties listed in "Setting Field Properties" on page 151, Digital Signature fields have the following properties:

List of Signed Fields The list of data fields that require an electronic signature. Separate fields with a semicolon (;) between them. All fields included in this list must be on the same form as the Digital Signature field.

File Attachment

Use the File Attachment class ID when the data field is actually a file you want users to attach. This data field provides menus in Process Express that allow end users to upload, download, and view files in a process. Users can use this field if they are accessing Process Express with Netscape Communicator 4.x, or Microsoft Internet Explorer 4.x or 5.x.

A File Attachment field appears in the HTML form as a signed applet. This applet attaches a file to the form. When the end user decides to attach a file to the form, the user clicks the icon for the applet and the applet uploads the file to a special location on the iPlanet Web Server. All files for a process instance are grouped in the same folder so you can locate and manage them easily.

You specify this location in the URL property of the Content Store. See "Setting Up the Content Store" on page 168 for more information. You specify what kind of file is uploaded in the data field's File Name property. For each kind of file you want users to be able to attach, you must create a separate data field.

Usage Tips for File Attachments

If you want to use file attachments with Secure Sockets Layer (SSL), your server must be SSL-enabled. To use SSL, end users must use Netscape Communicator to access Process Express. For information on how to enable SSL on your server, see the *iPlanet Web Server Administrator's Guide*.

If you have iPlanet Application Server installed on the same system as Process Express, you will not be able to use Netscape Communicator with file attachments. To avoid this situation, you can install iAS and Process Express on different systems, or you can use Microsoft Internet Explorer to access Process Express.

You must have the Web Publisher or Remote File Manipulation turned on in your iPlanet Web Server in order to use file attachments.

File Attachment Properties

In addition to the properties listed in "Setting Field Properties" on page 151, File Attachment fields have the following properties:

Alignment Defines the alignment of the file representation in the HTML form. Valid values are: CENTER, TEXTTOP, ABSCENTER, ABSBOTTOM. These position the file representation in the HTML form as centered, aligned with the top of the tallest text on the line, aligned with the center of the text, and aligned with the bottom of the text. The text may be moved to accommodate the alignment. Only one line of text is aligned with the data field. Subsequent lines of text appear below the data field. If no alignment value is specified, the alignment is handled by the browser.

Background Color The background color for the data field display area. Specify an HTML color code in the format #RRGGBB. For example, for pure blue, specify #0000FF. For more information about specifying colors with this format, consult any reference on HTML tags.

Border Width The thickness of the border to draw around the data field display area in pixels.

Debug Mode If this is set to true, the data field displays debug information in the browser's Java console.

Display Height The height in pixels of the data field display area. If you change this height, the size of the icon in the data field does not change. The display area gets more or less background space around the icon.

Display Width The width in pixels of the data field display area. If you change this width, the size of the icon in the data field does not change. The display area gets more or less background space around the icon. The optimal width is 100 pixels, but this depends on the size of the font and the value of the File name property.

File Name The file name which is displayed under the data field as a label in the HTML page. The extension of the File Name determines what kind of file can be uploaded via this file attachment data field. For example, if the File Name is attachment.html, then the data field can handle HTML files. If it is attachment.gif, then the data field can handle GIF files.

Font Name The font used for all text in the data field display area and also for the menu associated with the data field in Process Express.

Font Size The font size used for all text in the data field display area and also for the menu associated with the data field in Process Express.

Foreground Color The foreground color for the data field display area. The foreground color is used for text and other foreground elements. Specify an HTML color code in the format #RRGGBB. For example, for pure blue, specify #0000FF. For more information about specifying colors with this format, consult any reference on HTML tags.

Help URL A URL that points to a help page. If this property is defined, then when the data field is displayed in Process Express it has an additional menu item, User Help, that displays the page located at the Help URL. This URL must be an absolute URL. The intention here is that you can provide helpful information to your end users about the file to attach, such as what it is for or what kind of file it should be.

Icon Height The default height for the icon used in the data field display area. You cannot change this value.

Icon Width The default width for the icon used in the data field display area. You cannot change this value.

MAX URL Length The maximum length of the URL for files that are saved with this data field. It defaults to 100 characters. This URL is computed from the content store URL. Hence, this width should be at least (content_store_url) + 32 + length(File name) characters long.

Password

The Password class ID is used for password fields. When the user enters data into this field, or when it is viewable, the field value is shown as asterisks (*****).

In addition to the properties listed in "Setting Field Properties" on page 151, Password fields have the following properties:

Default Value Defines the default value of the field.

Display Size This required field defines the size of the text field in the HTML form.

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. Note that if this length is longer than the Display Size, some characters may not be displayed on the form. The Maximum Length must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Radio Buttons

The class ID Radio Buttons produces radio buttons, which appear on the end user forms as the button and the text next to them. This class ID is useful when the list of values is well known. You may also consider using the class ID Select List for this kind of information. Select List produces a drop-down list. See "Select List" on page 159 for more information.

In addition to the properties listed in "Setting Field Properties" on page 151, Radio Button fields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false.

Default Value Defines the default value of the field.

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. It must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Options A required field that represents the radio button options. To create them, type in all available values separated by semicolons (;). For example, to have a Yes and No radio buttons, in the Options text field, type:

```
Yes;No
```

Do not put spaces before or after the semicolon.

Select List

The Select List class ID produces a drop-down list. It useful when the list of values are well known. Another good class type for this kind of information is Radio Buttons, which produces radio buttons. See “Radio Buttons” on page 158 for more information.

In addition to the properties listed in “Setting Field Properties” on page 151, Select List fields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false.

Default Value Defines the default value of the field.

Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. It must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Options A required field that represents the values of the options in the drop-down list. To create the options, type in all values separated by a semicolon (;). For example, to have a Yes and No be the options in a drop-down list, in the Options text field type:

```
Yes;No
```

Do not put spaces before or after the semicolon.

TextArea

Use this class ID to create large areas where the user can enter text.

In addition to the properties listed in "Setting Field Properties" on page 151, text area fields have the following properties:

Default Value Defines the default value of the field.

Number of columns This required field defines the number of columns of this text area. The number of rows multiplied by the number of columns must be between 0 and 2000 (for LONGTEXT) or 0 and 255 (for TEXT).

Number of rows This required field defines the number of rows of this text area. The number of rows multiplied by the number of columns must be between 0 and 2000. The number of rows multiplied by the number of columns must be between 0 and 255 for the TEXT, and between 0 and 2000 for LONGTEXT.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

TextField

Use this class ID to create text fields. Use the Date and DateTime class IDs for dates and times, not TextField. For longer text areas, use the TextArea class ID

In addition to the properties listed in "Setting Field Properties" on page 151, TextFields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false

Default Value Defines the default value of the field.

Display Size This required property defines the size of the TextField in the HTML form.

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. Note that if this length is longer than the Display Size, some characters may not be displayed on the form. The Maximum Length must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

URL

A URL field is an input field in edit mode and a link in view mode. It is used for data that corresponds to a URL and is understood by a browser, for example the protocols http, ftp, ldap, etc.

In addition to the properties listed in "Setting Field Properties" on page 151, URL fields have the following properties:

Default Value Defines the default value of the field.

Display Size This required field defines the size of the text field in the HTML form.

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. Note that if this length is longer than the Display Size, some characters may not be displayed on the form. The Maximum Length must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Target This required field defines the window in which the URL is open in view mode. For example, NewWindow opens the URL in a new window. The Target property takes the same values as the TARGET attribute in HTML.

UserPicker Widget

UserPicker fields appear as an input field and an Address Book button on a form. The input field is where users enter the user information the field calls for. By using the Attribute property, you can specify what the user needs to put in the field (for example, distinguished name, user ID, email address and so on).

If the user already knows the information they want to put in the field, they type it in. If they don't know exactly what to type, they can search the corporate user directory by pressing the Address Book button. This button brings up a screen for searching the corporate user directory. Once the user finds the information they want, it appears in the input field. That information is stored in the database with the type of TEXT.

NOTE Because the values are automatically stored with a Data Type of TEXT, there is no Data Type property to specify for this class ID.

In addition to the properties listed in "Setting Field Properties" on page 151, UserPicker fields have the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false

Attribute the LDAP attribute you want to search on. Valid values include the distinguished name (dn), user ID (uid), common name (cn), email address (mail), department number, and manager.

Default Value Defines the default value of the field.

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. Note that if this length is longer than the Display Size, some characters may not be displayed on the form. The Maximum Length must be an integer between 0 and 255.

Custom Data Fields with Your Own Class ID

If the standard data field classes provided with iPlanet Process Manager do not fit your needs, you can create your own. For example, you can use custom data fields to access external data sources and to generate dynamic content at entry points. To use custom data fields you need to write Java classes that implement the field.

For example, you could write a database-driven Select List field such that all the options available to the end user are extracted from a database using an SQL query. Another example is a field that gathers database information from an external source

For more information about implementing your own custom fields, see the *Process Manager Programmer's Guide*.

Predefined Data Fields

The predefined data fields are templates that have been configured with default data that you can modify after you create the field. For example, the Address data field is a TextField with a default size. Select a template data field from the template list box, enter the name of the field, and click Create.

Process Builder contains the following predefined data fields:

- Address
- Name
- Telephone

Address

The predefined data field for addresses has the following predefined property values. You can change all but the field class ID.

Table 7-1 Predefined address properties

Property	**Value**
Data Type	Text
Display Name	Address
Field Class ID	TextArea
Short Description	Some Address

In addition to the properties listed in "Setting Field Properties" on page 151, Address fields contain the following properties:

Default Value Defines the default value of the field.

Display Size This required field defines the size of the text field in the HTML form.

Help Message A help message associated with a data field. The user sees it as text displayed in the bottom of the browser window (below the scroll bar).

Number of columns This required field defines the number of columns of this text area. The number of rows multiplied by the number of columns must be between 0 and 2000 (for LONTEXT) or 0-255 (for TEXT).

Number of rows This required field defines the number of rows of this text area. The number of rows multiplied by the number of columns must be between 0 and 2000. The number of rows multiplied by the number of columns must be between 0 and 255 for the TEXT, and between 0 and 2000 for LONGTEXT.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Name

The predefined data field for names has the following predefined property values. You can change all but the field class ID.

Table 7-2 Predefined name properties

Property	Value
Data Type	Text
Display Name	Name
Field Class ID	TextArea
Short Description	Some

Name fields contain the following properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false

Default Value Defines the default value of the field.

Display Size This required field defines the size of the text field in the HTML form.

Help Message A help message associated with a data field. The user sees it as text displayed in the bottom of the browser window (below the scroll bar).

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. Note that if this length is longer than the Display Size, some characters may not be displayed on the form. The Maximum Length must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Telephone

The predefined data field for telephone numbers has the following predefined property values. You can change all but the field class ID.

Table 7-3 Predefined telephone properties

Property	Value
Data Type	Text
Display Name	Telephone
Default Value	(XXX) XXX-XXXX
Field Class ID	TextField

Telephone fields contain the following additional properties:

Allow Search Specifies if the field can be used as a search criteria. Valid values are true or false

Display Size This required field defines the size of the text field in the HTML form.

Help Message A help message associated with a data field. The user sees it as text displayed in the bottom of the browser window (below the scroll bar).

Maximum Length A required property that defines the length of the field and the length of the database column that will be used to store values of that field. Note that if this length is longer than the Display Size, some characters may not be displayed on the form. The Maximum Length must be an integer between 0 and 2000. Use a value between 0 and 255 for the TEXT data type, and between 0 and 2000 for the LONGTEXT data type. If you want the field to be searchable, use the TEXT type.

On Blur The script that is run every time the field loses the focus within the HTML form. The script is client-side Javascript that is associated with the onBlur event handler of the field.

On Focus The script that is run every time the field gets the focus within the HTML form. The script is client-side Javascript that is associated with the onFocus event handler of the field.

On Value Change The script that is run every time the value of the field is changed. The script is client-side Javascript that is associated with the onChange event handler of the field.

Short Description A description of the field.

Deleting Data Fields

➤ **To delete a data field**

1. In the application tree view, select the field.
2. Right-click and choose Delete.

If you delete a field that you have already added to forms, you must also edit the forms to remove the field.

If the field is used by a field role, you must first delete the role, then delete the field.

Setting Up the Content Store

The content store is an HTTP URL where your application stores documents that are attached by File Attachment data fields.

Before deploying your application, if you have File Attachment data fields you need to set properties in your content store.

➤ **To set the properties of the content store**

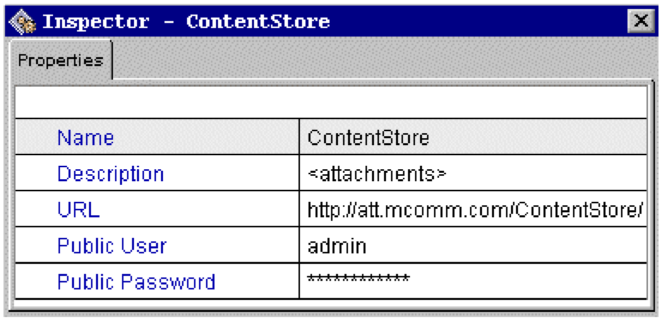
1. In the Application Tree View window, right-click the content store and choose Properties, or double-click content store, or highlight it and click Inspect.
2. Edit the properties.
3. Close the window.

   The changes are saved automatically.

The Content Store Inspector Window

Figure 7-2 shows the Inspector window for the content store.

Figure 7-2 Inspector window for the content store



Name	ContentStore
Description	<attachments>
URL	http://att.mcomm.com/ContentStore/
Public User	admin
Public Password	************

Name The name of the content store. You cannot change this property.

Description The description of the content store. This field does not appear in Process Express or Process Administrator.

URL The URL to the iPlanet Web Server document directory where you want to put the file attachments, such as `http://`*server_name*`/store`. The files for each process instance will be placed in the folder you specify. If the folder does not exist, iPlanet Process Manager creates it automatically when the application is deployed. Within the application folder, the documents (numbered by process instances) will be broken into subfolders, each able to contain a maximum of 64000 documents. For example, the first 64000 documents for the application go into a folder named part001, the second 64000 documents go into a folder named part002, and so on.

You can create the directory on the web server yourself if you prefer, rather than letting Process Manager create it for you at deployment time. For details, see "Step 4: Optionally, Add a Directory to the Web Server" on page 296 in Chapter 14, "The Data Sheet Application."

Public User The user name used to access the URL where the file attachments are stored. This user must be defined within the web server, and must have all access permissions for the directory where the documents are stored.

Public Password The password for the web server user defined in the Public User property.

Troubleshooting the Content Store

If users have trouble attaching files in their applications, it could be a problem with access control on the content store. The web server that hosts the content store must have its access permissions set so that the public user can write to it. If you are using a Unix system, you should also check to make sure that the Unix permissions on the folders are set so that the web server public user has write access. The web server error log may also help you diagnose content store errors.

For details about configuring the content store for use with a secure web server, see "Enabling Content Store Over https" on page 48.

Chapter 8

Designing Forms

This chapter explains how to define forms in Process Builder, and how to use the HTML editing features.

This chapter includes these sections:

Planning Forms

As part of your application, you design the forms that the users are going to use to complete work items or monitor the process. You can assign a custom form for each activity or reuse the same form at multiple activities. You can create separate forms for the people participating in the process and the people monitoring the process. The forms define the data the users get to see and how they can interact with the data.

Before you create a form, ask yourself the following questions:

- What data field should be included in the form?
- Should the data be editable, viewable, or hidden? If a field is designated as editable, only the assignee can edit it. It will be viewable to others.
- Is the form for editing data or monitoring the process?
- How should the form look, and where should the data appear in the form?

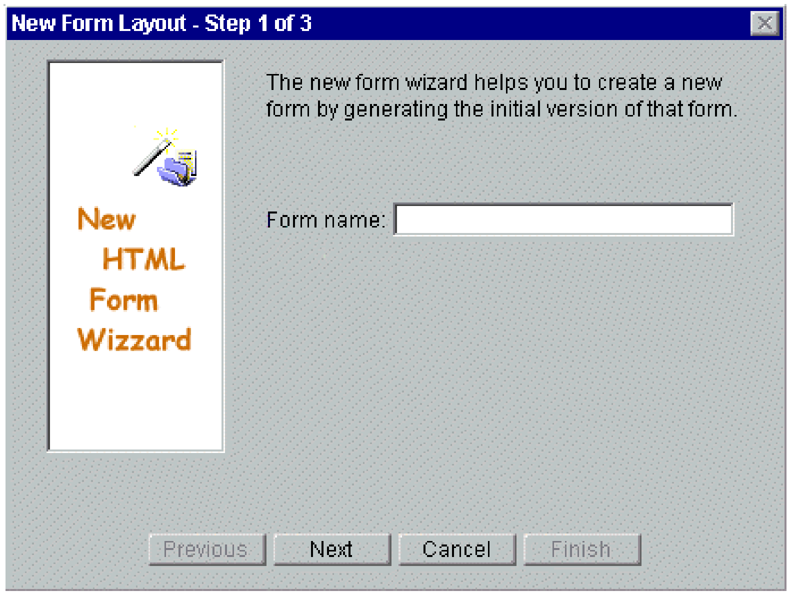
Creating Forms

➤ **To create a form**

1. From the Insert menu, choose Form.

   The New HTML Form Wizard appears, as shown in Figure 8-1.

Figure 8-1 New Form Layout, Step 1



The new form wizard helps you to create a new form by generating the initial version of that form.

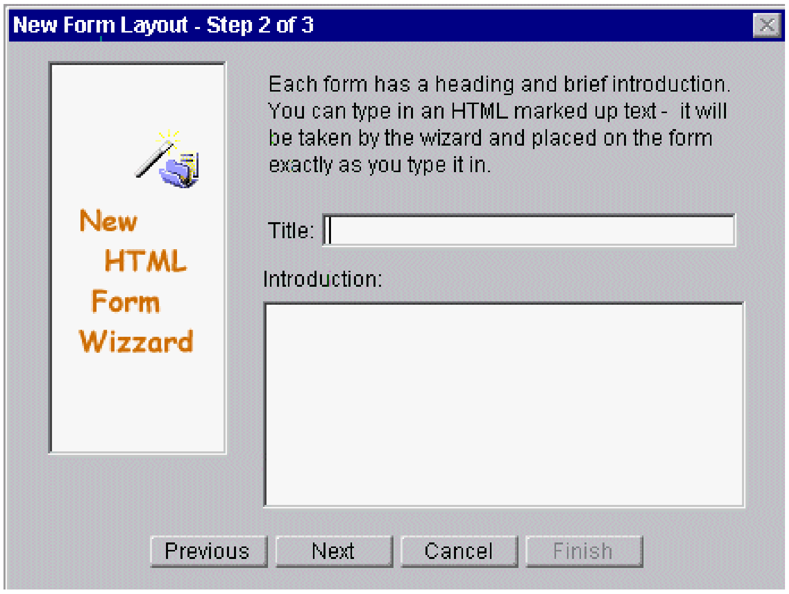
2. Enter a name for your form.

   Because the form contains HTML, the name typically ends in .html. The name must not exceed 36 alphanumeric characters, including the file extension.

3. Click Next.

   Page 2 of the wizard appears, as shown in Figure 8-2.

Figure 8-2 New Form Layout, Step 2

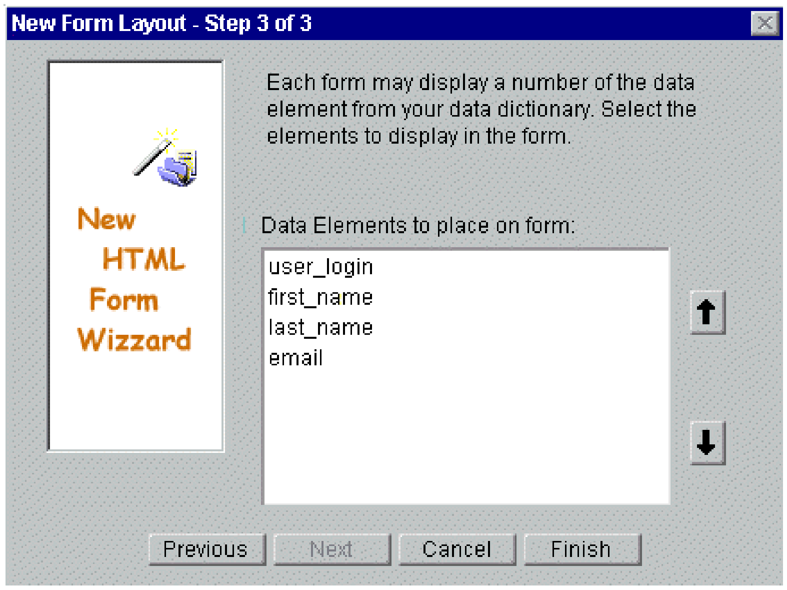


Each form has a heading and brief introduction. You can type in an HTML marked up text - it will be taken by the wizard and placed on the form exactly as you type it in.

4. Enter the title that you want to have appear on the top of the form. Enter some introductory text that explains the form to the users.
5. Click Next.

   Page 3 of the wizard appears, as shown in Figure 8-3.

Figure 8-3 New Form Layout, Step 3



Each form may display a number of the data element from your data dictionary. Select the elements to display in the form.

Data Elements to place on form:

user_login
first_name
last_name
email

6. A list of available fields appears, listed in the order they will appear on the form. Highlight the field you want to add to the form.

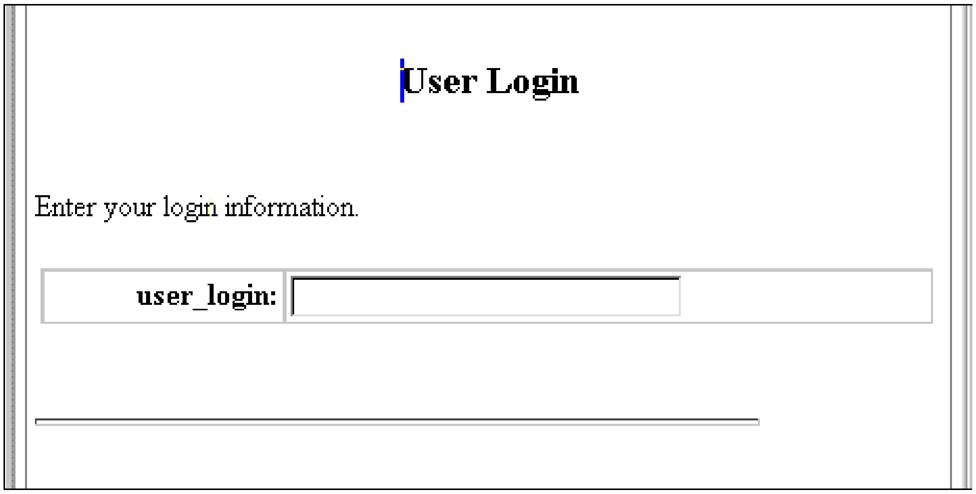
    - Click the up and down arrow keys to change the placement order of fields on the form.

    - To select multiple data fields, press and hold the mouse, and drag it across the additional field names.

7. Click Finish.

    The form opens in the form editor for additional editing, as shown in Figure 8-4.

Figure 8-4 Form created by New Form Layout wizard


User Login

Enter your login information.

user_login:

8. To make additional changes, edit the form using the HTML editor tools provided in the toolbar.

   See "Using the HTML Page Editor Toolbar" on page 177 for an explanation of these tools.

9. Add more data fields by dragging them from the Data Dictionary (in the application tree view) to the form.

10. Set the display mode of the field to edit, view, or hidden.

    For more information, see "Changing Field Properties for a Form" on page 180.

11. When you are finished, close the form window and save it.

In addition to the information you create for a form, Process Manager automatically adds two form elements: a header that describes the work item to the end user, and an action bar at the end of the form. These elements appear when the user sees the form in Process Express. The action bar contains two items: the actions included in the activity, and buttons showing the text of the transitions between actions.

If you have designated the activity as "Allow to Add Comment," the form also displays a text field for users to add their comments to a process instance. The comments are included in the process instance history.

Modifying Forms

You can modify forms either in Process Builder's form editing tool or in an external editor.

Using Process Builder's Form Editor

To modify a form, double-click its name in the application tree view. The form appears in the Page Editor. Each form appears in its own Page Editor window.

You can edit the text on the form using the HTML editor toolbar. You can copy text and fields from one form and paste them in another. You can also use your right mouse button to bring up a menu of editing commands.

Using an External Editor

You can use an external HTML editor to edit the layout of Process Manager forms for your application with the following cautions:

- You must not change any HTML tags or attributes that you or your editor do not understand. Process Manager data fields are implemented as customized SERVER tags. Some HTML authoring tools will delete tags and attributes that are not standard HTML.
- You must leave these tags in all caps: HEAD, BODY, HTML, and SERVER.
- If you use dynamic HTML or JavaScript in a form created with an external HTML editor, you may find that Process Builder corrupts them and you have to use your external editor to clean up the form.

If you edit the forms in an external editor, be sure not to modify any of the customized SERVER tags, or the data fields on the forms will not work.

If you edit a web page for a form while it is also open in Process Builder, be sure to click on the form again in Process Builder before saving the application. When you click on a form that has been edited externally, Process Builder asks if you want to load the modified form with the latest changes.

Note that the web page for the form only contain a portion of the information the end-user sees on the form. The Process Engine inserts the banner, buttons, and comment areas of the form, as well as the <FORM> tag itself.

The HTML forms for each application are stored in the folder on your local file system where the application is stored.

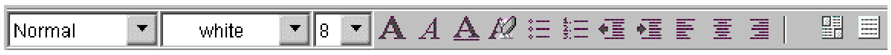
➤ **To edit Process Manager forms using an external HTML editor**

1. Create a form using Process Builder (see “Creating Forms” on page 172) or use a form already created in an external HTML editor.
2. Use Process Builder to add data fields. When you are finished, close the form window and save it.
3. Open the form in your HTML editor.
4. Make and save your changes.

Using the HTML Page Editor Toolbar

The HTML Page Editor toolbar, shown in Figure 8-5, lets you edit and format the HTML forms that you create.

Figure 8-5 HTML Page Editor Toolbar



The drop-down lists and icons assign HTML formatting to the text you choose. You can point your mouse over a toolbar icon to see a “tool tip,” or brief description, of these items. In addition, the drop-down lists and icons are described as follows, from left to right:

- The Style drop-down list lets you format text as normal text, HTML headings, or HTML list items.
- The next two drop-down lists let you choose a color and font size for the text.
- The next three icons (letter A) make text bold, italic, or underlined. The icon following it removes changes to the font size or font style (bold, italic, or underline).
- To the right of the style icons are several formatting icons for creating lists, changing indentation, and changing alignment.
- The last two icons display the form in layout mode (the default) or as HTML source. Layout mode displays the form approximately as you would see it in a browser, with all elements laid out. The source view shows the HTML tags. In this view, you can click a “Reformat Source” button. This makes the HTML code more readable by automatically inserting line breaks as appropriate.

Using the Edit, Insert and Format Menus

The menus at the top of Process Builder also contain commands you can use to modify forms. Since the keyboard shortcuts supported by your operating system may not be available, these menus provide you with editing functionality.

Edit

Use the following Edit Menu items to edit your HTML forms.

Cut Cuts an HTML element from a form.

Copy Copies an HTML element on a form.

Paste Pastes an HTML element into a form.

Delete Deletes the highlighted element or text from the form.

Insert HTML Element

The item on this menu that applies to editing HTML page is HTML Element. You can choose to insert an item from the submenu:

Link Adds an HTML link. To edit the link, use your right mouse button. When you deploy an application, the structure remains the same as in the local Applications folder, so you can use relative links.

Image Adds an image to the HTML page. You can double-click on the image to get a property sheet where you can fill out the image name, source, etc. The image must already be in the application directory, or you can import it using the Import command from the file menu. When you deploy an application, the folder structure remains the same as in the local Applications folder, so you can use relative paths when you import images.

Horizontal Line Adds a horizontal line to the HTML page.

Table (table, row, column, cell) Lets you add a new table (if you are not editing an existing table), or add a row, column, or cell to an existing table.

Client JavaScript Adds client-side JavaScript to the form. Double-click the Script icon in the form, or use your right-mouse button to get to the properties window. In the properties window, click the ellipsis (...) next to the JavaScript Code property to access the editor window. Type your JavaScript there.

Format

These are standard HTML editing commands. Use them to edit forms.

Size Applies the font size you select to the text.

Style Formats text as either bold, italic, underline, or strikethrough.

Remove all styles Removes size and style formatting from text.

Heading Applies an HTML heading tag to the text.

List Applies an HTML list tag to the text. You can choose either a bulleted or a numbered list.

Align Aligns text at the left, right, or center of the page.

Decrease Indent Decreases the amount of indention.

Increase Indent Increases the amount of indention.

Using Right-Mouse-Button Menu Commands

When you edit your HTML forms, your right mouse button lets you perform commands.

When you right-click on a data field on a form, you get a pop-up menu with these commands:

Cut Cuts selected object.

Copy Copies selected object.

Paste Pastes previously copied or cut object.

Table Displays extra commands for HTML tables, including commands for cutting, copying, and pasting; for deleting rows, columns and cells; and for setting table, row, and cell properties.

Data Dictionary Displays the data field's inspector window so that you can set any properties. Note that this changes the field's properties on all forms that use it. This option is the same as double-clicking the field in the application tree view.

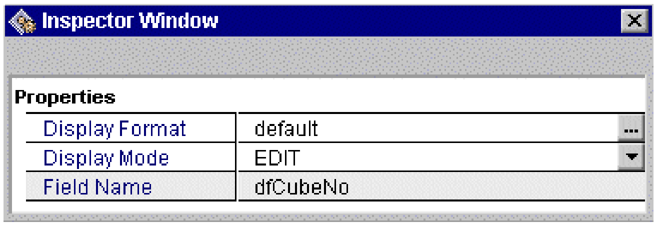
Properties Accesses properties of the field on the form. It allows you to change its display mode: EDIT, VIEW, and HIDDEN. It also shows the Field name and the ID of the component. It lists events and methods associated with the field.

Changing Field Properties for a Form

After adding a field to a form, you can change the field's properties on that form. These changes affect only the field on the form; they do not affect other occurrences of the field within the application.

To display the field properties for that form, double-click the field on the form. The Inspector window appears, as shown in Figure 8-6:

Figure 8-6 Field properties on a form



Display Format	default
Display Mode	EDIT
Field Name	dfCubeNo

Display Format The display format of the field. This property is unused for built-in fields. For custom fields that you have implemented through your own Java classes, you can optionally define a display format. Typically, the "default" setting is sufficient.

Display Mode The drop-down list contains the following values: VIEW, EDIT, and HIDDEN.

- In view mode, a user can see the field but cannot change it.
- In edit mode, users can edit the field if they are the assignee for that work item. For other users, an edit field appears as a view field.
- In hidden mode, the user cannot see the field. However, the value of the hidden field can be accessed by other fields in the form or by client-side JavaScript scripts. One use for the hidden mode is a signature field, which retrieves the values of the fields that need to be signed and presents these values to the user for signature.

Field Name The name of the field, which you cannot change.

Hints for Setting Field Properties

As you change the display mode, the form updates itself to show the correct visual mode for the field. Editable fields are in boxes on a white background; view-only and hidden fields are on a gray background.

Note that in Process Builder, hidden fields appear on a form. This lets process designers change the mode. It is easier to edit a form if you put all hidden fields at the end of the document. In this way, your view of the form is as close as possible to the user's view.

If you define the same field as editable on more than one form, the value that is stored in the database at the end of the process instance is the last value entered.

Computed fields are always hidden from the user, even in view mode. If you want to display a computed field's information to the user, use a text field instead.

Accessing the Data Dictionary

By right-clicking a field in the form, you get a drop-down list that has a Data Dictionary option. If you choose the Data Dictionary option, you access the properties of the field.

CAUTION The properties you edit in the data dictionary are shared by all instances of the field. If you change the properties of a field, you change all occurrences of a field on all forms.

Using Scripts to Validate User Input

You can use client-side JavaScript to validate user input on a form as the user submits it. To validate this input, write a client-side JavaScript script called `onSubmitForm()` and insert it into the form. This script is invoked just before the form is submitted.

- If the input is not valid, the script returns `false`, and the form input is not submitted.
- If the input is valid, the script returns `true`, and the form input is submitted. If the `onSubmitForm` function returns `false`, the form is not submitted.

For every data field represented in a form, the form contains a form element that has exactly the same name as the data field. Client-side scripts such as `onSubmit` functions can access data field values as they are currently displayed in the form. The scripts do this by accessing form elements that have the same name as the data field.

See the section "onSubmitForm Example" on page 233 for a coded example of an `onSubmitForm` function.

Adding a Banner to Forms

When an end user sees a form, a banner is included at the top.

➤ **To create a banner**

1. Create a file to be your banner and name it `banner.gif`.
2. In Process Builder, open the application to which you want to add the banner.
3. From the File menu, choose Import.
4. Click Select and navigate to the `banner.gif` file to import.
5. Enter "images" in the "into folder" field.
6. Click Import.

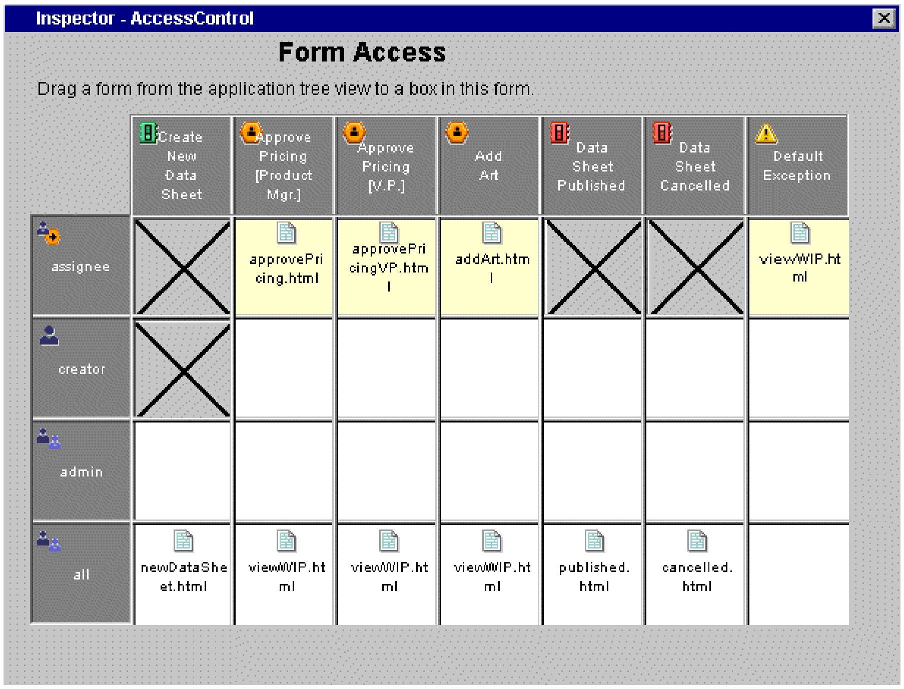
   The `banner.gif` file is imported into the images folder. If the images folder does not already exist, Process Builder creates it.

NOTE The file must be named `banner.gif` and must be in the folder `builder/Applications/`*app_name*`/images/banner.gif`.

Setting Access to Forms

Form Access associates forms with specific users and activities at certain steps in the process. To set access control for your forms, click Form Access, which is located near the bottom of the application tree view. As a result, you see the Form Access window. This window displays form access information in a table, as shown in Figure 8-7.

Figure 8-7 Form Access window



Inspector - AccessControl

Form Access

Drag a form from the application tree view to a box in this form.

	Create New Data Sheet	Approve Pricing [Product Mgr.]	Approve Pricing [V.P.]	Add Art	Data Sheet Published	Data Sheet Cancelled	Default Exception
assignee	✗	approvePricing.html	approvePricingVP.html	addArt.html	✗	✗	viewWIP.html
creator	✗						
admin							
all	newDataSheet.html	viewWIP.html	viewWIP.html	viewWIP.html	published.html	cancelled.html	

The row at the top represents steps in the application. The kind of step involved is indicated by an icon. The column on the side represents the groups and roles used by the application. The groups and roles are in the priority order you set when defining groups and roles.

To assign a form to a group or role at a process step, drag a form from the application tree view into the appropriate square in the process map. If you drag a form to the title box of an activity (top row), it shows up in every square in that column that does not already contain a form. If you drag a form to a role/group name box on the left, the form shows up in every square in the row that does not already contain a form. You cannot drag a form to a box that has an X through it.

At each step in the process, the application looks from top to bottom for the appropriate form to present to the user.

For example, in Figure 8-7, in the "Create New Data Sheet" step, the application presents the same form to everyone (all), regardless of their role in the process. Anyone can initiate a datasheet. However, at the "Approve Pricing" step, the assignee (the product manager who needs to approve the pricing) sees a form for approving pricing. Everyone else is only able to see a form that gives read-only information about the work in process.

To remove a form from the table, click the square the form is in and press the delete key.

Forms for Assignees

The forms for the assignees must contain all the information, both editable and view-only, that assignees need in order to complete a step. You must have a special form for the assignee for every activity. Because entry and exit points do not have work items, they do not require an assignee form. These boxes are crossed out, indicating that you cannot add a form to them. For example, in Figure 8-7 you can see that there is a special form for the assignee in all steps except for entry and exit points.

Forms for Monitoring the Process

Often people who are not necessarily participants in the process may want to monitor its progress. In those cases you might create a special form for all interested parties and assign it to the "all" group, or another group that contains people interested in the process.

In other cases, you do not want to give a nonparticipant view because the process involves sensitive information. For example, in an employee time off request application, an employee's vacation request is private, and should not be available for others to view. In these cases you restrict access by only assigning forms for the people who should have access to the information, for example the assignee and the creator.

Forms for the Administrator

The administrator can perform the following functions on a process instance:

- delegate the work item to a different user
- extend the expiration date of the work item
- change the state of a process instance
- move the work item to a different activity

In order to perform these functions, the administrator must belong to an application's "admin" group. Usually, you make sure the "admin" group has access to a form at each step of the process, so that the administrators have access to the field information they need to make changes to work items. The form can be either a special form for the "admin" group, or a form for the "all" group.

Setting Access to an Entry Point

For many processes, all employees in a company need to be able to initiate a process instance. For example, every employee needs to be able to submit a time off request. In these cases you set up a form for initiating a process instance and assign it to the "all" group (as shown in Figure 8-7).

In other cases you do not want all users to be able to initiate a process instance. For example, if you had a web site that contained the marketing material for a product suite, you might want only people from the marketing department to be able to add documents to the web site. In that case you might make the form for initiating a process instance available only for a "marketing" group.

Please note that you cannot set a form for the creator at the entry point, since the creator role is not used until after the entry point.

Chapter 9

Using Scripts

This chapter describes how to use the scripts available in Process Builder. It also briefly discusses creating your own scripts.

This chapter includes these sections:

Overview of Scripts

By using scripts, you can customize and configure your application's activities, automated activities, entry points, exit points, decision points, and transitions. For example, scripts let you assign users to activities and set expiration dates of activities.

Kinds of Scripts

Some scripts are provided with Process Builder, but you can also create your own. All scripts for an application are stored in the Script Dictionary in the application tree view. Table 9-1 describes the available scripts. Each type has its own purpose and its own return value.

NOTE Each type of script has its own folder within the Script Dictionary. You can drag scripts from one folder to another, but you must update the script so that it returns the data required by the new type before the script will work.

Table 9-1 Available JavaScript scripts

Kind of Script	Purpose	Return Value
Assignment	Assigns a user or users to perform a work item.	Returns an array containing the user ids and/or group names of the users and/or groups to whom this work item is assigned.
Automation	Performs an automated activity. It is executed when the process reaches the automated activity.	Returns `true` if the activity is successful, otherwise returns `false`.
Verification	Executes when a work item is completed.	Returns true if the activity is successfully verified, otherwise returns false.
Completion	Executes when a work item is completed. This script runs after the verification script.	Returns `true` if the activity is successful, otherwise returns `false`.
Expiration Setter	Sets the expiration date of a work item.	Returns a Date object.
Expiration Handler	Runs when a task has not been completed before its expiration date is reached.	Returns `true` if the expiration is successfully handled, otherwise `false`.
Email Address in Notification	Specifies an email address to which a notification will be sent, either in the To: line or Cc: line.	Returns a valid email address, or a comma-delimited list of valid email addresses. Values can be passed as a string or as a JavaScript array.

Table 9-1 Available JavaScript scripts

Kind of Script	Purpose	Return Value
Email Subject in Notification	Specifies the subject of an email message.	Returns a string specifying the subject of the email message.
Email Body in Notification	Specifies the body content of an email message.	Returns a string specifying the body of the email message.
Decision point or automation script condition	Specifies a condition that is used to determine which path to follow in the process.	Returns `true` if the condition is met otherwise `false`
Initialization script	Specifies tasks to perform when the *application* first starts.	Returns `true` if the initialization is successful, otherwise `false`
Shutdown script	Specifies tasks to perform when the *application* shuts down.	Returns `true` if the shutdown is successful, otherwise `false`
Toolkit script	A library function. It can be called by other functions or used as the value of any script for which it returns the appropriate kind of value.	Returns whatever is appropriate.

When to Use Scripts

To edit a script, double-click it in the application tree view, or highlight it and click Inspector. The script text appears in a text edit window. You can also edit scripts using other tools by opening the files where they are stored on your local machine (usually in the `builder/Applications` folder).

Table 9-2 shows what kinds of scripts are available for you to use at each process step.

Table 9-2 Script Types by Process Step

Script Type	Entry Point	User Activity	Automated Activity
Assignment		Available	
Automation			Available
Expiration setter		Available	
Expiration handler		Available	
Verification		Available	Available
Completion	Available	Available	Available

About Writing Scripts

You can define your own server-side scripts for any situation where scripts are used. The scripts must be written in JavaScript. You can write them in the script editor in Process Builder. Or you can write them in the editor of your choice, but you must later import them into Process Builder.

Scripts can use any core JavaScript language and objects, such as `array` or `date`. Scripts can also use additional objects that are provided by Process Manager. These objects include `processInstance`, `workItem`, `corporateDirectory` and `contentStore`. These scripts cannot use client-side objects such as `document` and `window`.

For more information about writing scripts, see Chapter 11, "Advanced Techniques for Scripting."

Predefined Scripts

Process Manager provides a set of predefined server-side scripts, all of which can be accessed from Process Builder. The scripts fall under the following categories:

- "Assignment Scripts"
- "Completion Scripts"
- "Email Notification Scripts"
- "Initialization and Shutdown Scripts"

Assignment Scripts

Table 9-3 lists the predefined assignment scripts that are available in Process Builder. The documentation below explains what they do, and what parameters you need to supply when you use them to assign a user to an activity. See "Setting Activity Assignments" on page 102 for more information on using assignment scripts in activities. For details on creating assignment scripts, see Appendix A, "JavaScript API Reference."

Table 9-3 Assignment Scripts

Script	Description
`randomToGroup`	This script selects a user randomly from the users that belong to the group specified by `groupname`. The parameter to edit is `groupname`, which is the name of the group that is defined in the Groups and Roles folder. It can be an application group, a corporate group, or a dynamic group.
toCreator	This script returns a JavaScript array user ID of the user who created the process instance. `toCreator` is the default assignment script for a user activity. This script accepts no parameters.
`toGroup`	This script returns the qualified role name of the group name specified by the `groupName` parameter. If this parameter does not correspond to an actual group in the application's Groups and Roles folder, this script returns `null`.
`toManagerOfCreator`	This script assigns the work item to the manager of the user who created the process instance. The manager relationship is based on the manager entry of the user in the corporate user directory. For this script to return the manager of the user, the manager attribute of the user in the corporate user directory must contain the distinguished name (DN) of the manager. No parameters need to be edited in this script.
`toManagerOfRole`	This script assigns the work item to the manager of the person defined in a field role. The manager relationship is based on the manager entry of the user in the corporate user directory. For this script to return the manager of the user, the manager attribute of the user in the corporate user directory must contain the distinguished name (dn) of the manager. The parameter to edit is `rolename`, which is the name of the field role defined in Groups and Roles folder.

Table 9-3 Assignment Scripts *(Continued)*

Script	Description
`toManagerOf`	This script assigns the work item to the manager of the user specified by its user ID. The manager relationship is based on the manager entry of the user in the corporate user directory. For this script to return the manager of the user, the manager attribute of the corporate user directory entry of the user must contain the distinguished name (dn) of the manager. The parameter to edit is `userID`, which is the user ID of the person whose manager you want to assign an activity to.
`toUserById`	This script assigns the work item to a user based on the user ID. The parameter to edit is `userID`, which is the user ID of the person to whom you want to assign the activity.
`toUserFromField`	This script assigns the work item to the user whose user ID is stored in the `datafieldname` field of the process instance data. The DN of the user is stored in this field. If you have a field role, using this script has the same affect as assigning the work item to the field role on the Assignment Selection dialog box using the "A group or role" radio button. The parameter to edit is `datafieldname`, which specifies the name field where the user ID has been stored. This field has to exist in the application data dictionary and has to contain the user ID of the user the work item should be assigned to.
`toParallelApproval`	This script assigns the activity to all the people specified in `arrayOfUserIDs`, and tracks if they approve or reject the item they are approving. You also need to use the `checkParallelApproval` completion script. The parameters to edit are `arrayOfUserIDs` and `FieldName`. The `arrayOfUserIDs` is a JavaScript array which contains the user IDs of the people who need to perform the approval. The `FieldName` is the field that keeps track of who has performed the approval and who still needs to do so. The field is a computed field of length 2000 that you have to add to the data dictionary.

Completion Scripts

There is one predefined completion script, `checkParallelApproval`. It is used for parallel approval.

This script runs when the parallel approval activity is completed. If any user chooses the "stopper action" (that is, refuses to approve the item) the completion script performs the appropriate action. If all users complete the activity without choosing the stopper action (that is, all approve the item) this script performs the appropriate action.

The parameters are `FieldName` and `labelOfStopperAction`. The `FieldName` is the field that keeps track of who has performed the approval and who still needs to do so. The field is a computed field of length 2000 that you have to add to the data dictionary. The `labelOfStopperAction` is the name of the action that a user can take that stops the approval.

For more information on parallel approval, see "Using Parallel Approval" on page 103.

Verification Scripts

There are no predefined verification scripts.

Email Notification Scripts

This section describes the predefined email notification scripts:

- `defaultNotificationHeader()`
- `defaultNotificationSubject()`
- `emailById(userId)`
- `emailOfAssignees()`
- `emailOfCreator()`
- `emailOfRole(role)`

For examples of using these scripts, see the *Process Manager Programmer's Guide*.

defaultNotificationHeader()

Returns the default notification header for a notification message body. The header contains information about the current work item, such as the current activity name, the process instance ID, the creation date of the process instance and the expiration date (if any).

This function may be used as the notification body script by itself, or may be embedded in your own notification body script. You may also use this function from a template evaluated using `evaluateTemplate()`. The function may only be used successfully from a script associated with a notification; if used anywhere else, an empty string is returned.

The text returned from this function will depend upon the content-type of the notification. If the content-type is `text/html`, the header will be a series of HTML tags; if the content-type is `text/plain`, the header will be plain text.

defaultNotificationSubject()

Returns the default notification subject for the notification subject line. The subject contains information about the current process instance, such as the process instance ID the priority and the title string.

This function may be used as the notification subject script by itself, or may be embedded in your own notification subject script. The function may only be used successfully from a script associated with a notification; if used anywhere else, an empty string is returned.

emailById(userId)

Returns a string of comma-delimited email addresses for the user with the given user ID. The `mail` attribute for the user must contain a valid email address in the corporate user directory. If the `mail` attribute of the user does not have a value, this function logs an error and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

emailOfAssignees()

Returns a string of comma-delimited email addresses for all the assignees of the work item. The `mail` attribute for each assignee must contain a valid email address in the corporate user directory. If the `mail` attribute is empty for any assignee, no address is added to the string for that assignee. If no assignee has a value in their mail attribute, the function logs an error message and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

emailOfCreator()

Returns a string of the email address of the user who created the process instance. The user's `mail` attribute must contain a valid email address in the corporate user directory. If the `mail` attribute of the user does not have a value, this function logs an error and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

emailOfRole(role)

Returns a string of the email address of the user performing the given role. The user's `mail` attribute must contain a valid email address in the corporate user directory. If the `mail` attribute of the user does not have a value, this function logs an error and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

Initialization and Shutdown Scripts

In addition to the predefined assignment scripts, every new application you create has an initialization script and a shutdown script in the script dictionary. The initialization script is called when the application is initialized, and the shutdown script is called when the application stops.

You can use the initialization script to set variables and create objects that are needed for the duration of the application. You use the shutdown script to release any resources allocated by the initialization script, and perform any final cleanup needed.

Creating Scripts

In addition to using the predefined scripts described in this chapter, you can also create your own scripts.

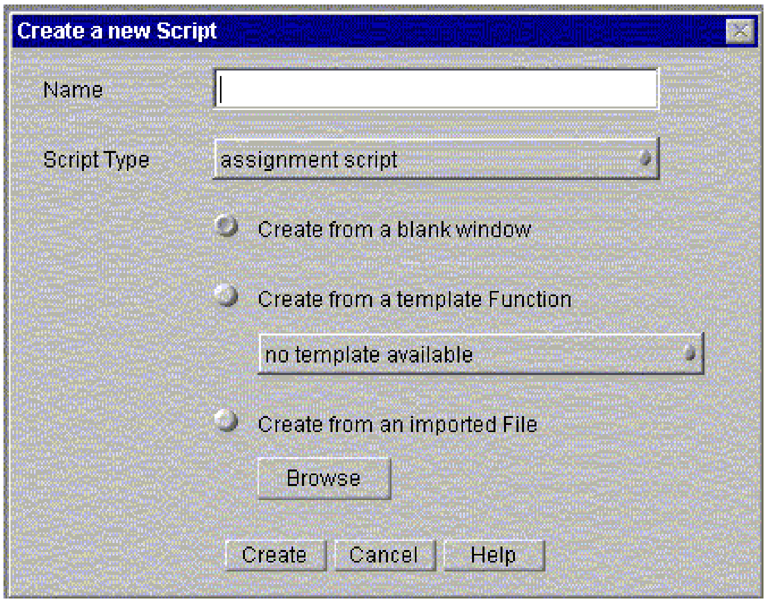
➤ **To create a script**

1. From the Insert menu, choose Script. The Create a New Script dialog box appears, as shown in Figure 9-1.
2. Name the script. The name must contain only alphanumeric characters.

3. Choose the type of script you want to create:

- Assignment Script
- Automation Script
- Expiration Setter Script
- Expiration Handler Script (also known as an Expire script or an On Expire script)
- Verification Script
- Completion Script
- Toolkit Script

Figure 9-1 The Create a New Script dialog box


Create a new Script

Name

Script Type: assignment script

Create from a blank window
Create from a template Function
no template available
Create from an imported File
Browse
Create
Cancel
Help

4. Click a radio button to choose how you want to create the script.

- "Create from a blank window" lets you type your script into a blank window, without basing it on an existing script. If you choose this radio button, go to Step 5.
- "Create from a template function" lets you base your script on another script of the same type which you have previously designated as a template. If you choose this radio button, choose a template from the drop-down list, and go to Step 5.
- "Create from an imported file" lets you import an existing script from outside the application, or copy one from within the application. If you choose a script from outside the application, you can save the script with the same name. If you choose a script from the current application, you must give the script a unique name. Click Browse to navigate to the `.js` file you want, and then click Open. Then go to Step 5.

5. Click Create. Your new script appears in the Script Editor Window.

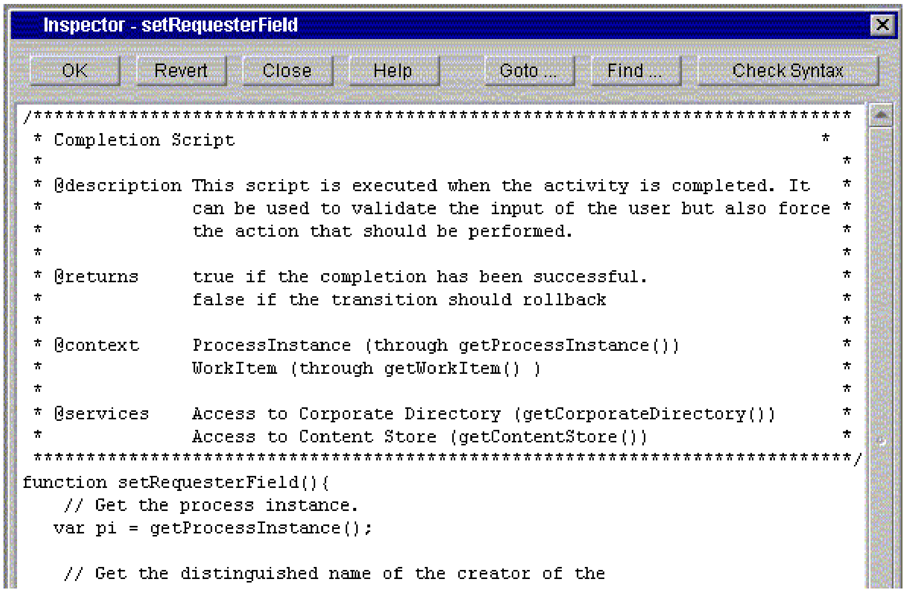
NOTE The `.js` files you import must follow the conventions for `.js` files in Process Manager. The name of the file must be the same as the name of the script, and each file can contain only one script.

The Script Editor Window

The script editor window appears when you are creating a new script or editing an existing script. If you are creating a new script from a blank window, the script editor appears with comments in the window pertaining to the type of script you are creating.

The script editor has a text area for typing in your script, and several buttons at the top, as shown in Figure 9-2.

Figure 9-2 A portion of the script editor window



```
/***************************************************************************
 * Completion Script                                                     *
 *                                                                         *
 * @description This script is executed when the activity is completed. It  *
 *              can be used to validate the input of the user but also force *
 *              the action that should be performed.                       *
 *                                                                         *
 * @returns     true if the completion has been successful.                *
 *              false if the transition should rollback                    *
 *                                                                         *
 * @context     ProcessInstance (through getProcessInstance())             *
 *              WorkItem (through getWorkItem() )                          *
 *                                                                         *
 * @services    Access to Corporate Directory (getCorporateDirectory())    *
 *              Access to Content Store (getContentStore())                *
 ***************************************************************************/
function setRequesterField(){
    // Get the process instance.
   var pi = getProcessInstance();

    // Get the distinguished name of the creator of the
```

OK Saves changes to the script and closes the window.

Revert Rolls back any changes you've made since the last time you saved the script. The window remains open.

Close Closes the window without saving changes.

Help Launches a web page containing help about this script.

Goto Lets you specify a line number to go to in the script.

Find Lets you specify a text string to search for in the script.

Check Syntax Checks the syntax of the script you've created, and displays an error message if there is an error.

Setting a Script as a Template

If you want to use a script as the basis of future scripts within the current application or other applications, you need to designate it as a template.

➤ **To set a script as a template**

1. In the application tree view, right click on the script name.
2. From the menu, choose “Set as template”.

The script is added to the drop-down list of templates that appears when you insert a new script. You can only use a template to create the same type of script. For example, an assignment script template only appears in the drop-down list if you are creating a new assignment script.

If you choose to create a script based on a template, the new script must have a different name from the template script.

When inserting a new script, if you choose to define it from a template function, the body of the template function will be used as the body of the new script. You can then modify the new script to suit your needs.

For example, suppose you define a function called `checkPageCount()` that checks if the value of the `pageCount` field is a number. Then you set it as a template function. Next time you need a script to check that the value of a different field is a number, you could create the new script from the `checkPageCount()` template script, and replace references to `pageCount` by the desired field.

Using Client-side Scripts

Most of this chapter describes server-side scripts that run at various stages in the life cycle of a process instance. However, you may also need to know something about client-side scripts. This section describes information that you may find useful.

Process Builder supports the following client-side scripts:

- Embedded client-side scripts. These are attached by using Process Builder’s Insert menu, then the HTML element submenu, and then choosing the Client JavaScript menu command.
- Event handlers for form elements. Event handlers include scripts such as `onValueChange` and `onClick`.

Client-side scripts run in the web page and can access other objects, such as form elements, in the current page. Client-side scripts can use the standard JavaScript language and client-side objects such as `form`, `window` and `document` that are available to all client-side JavaScript scripts.

Every web page displaying a work item contains a single form, which client-side scripts can access using `forms[0]`. For example, the following statement sets the variable `budget` to the value currently showing in the form element named `budget`:

```
var budget = document.forms[0].budget.value;
```

Every data field represented on a form has an associated form element of the same name. For example, if the `AuthorName` field is represented on the form, then the form contains a form element whose name is `AuthorName`. In some cases, this form element is hidden, but in all cases, such a form element exists.

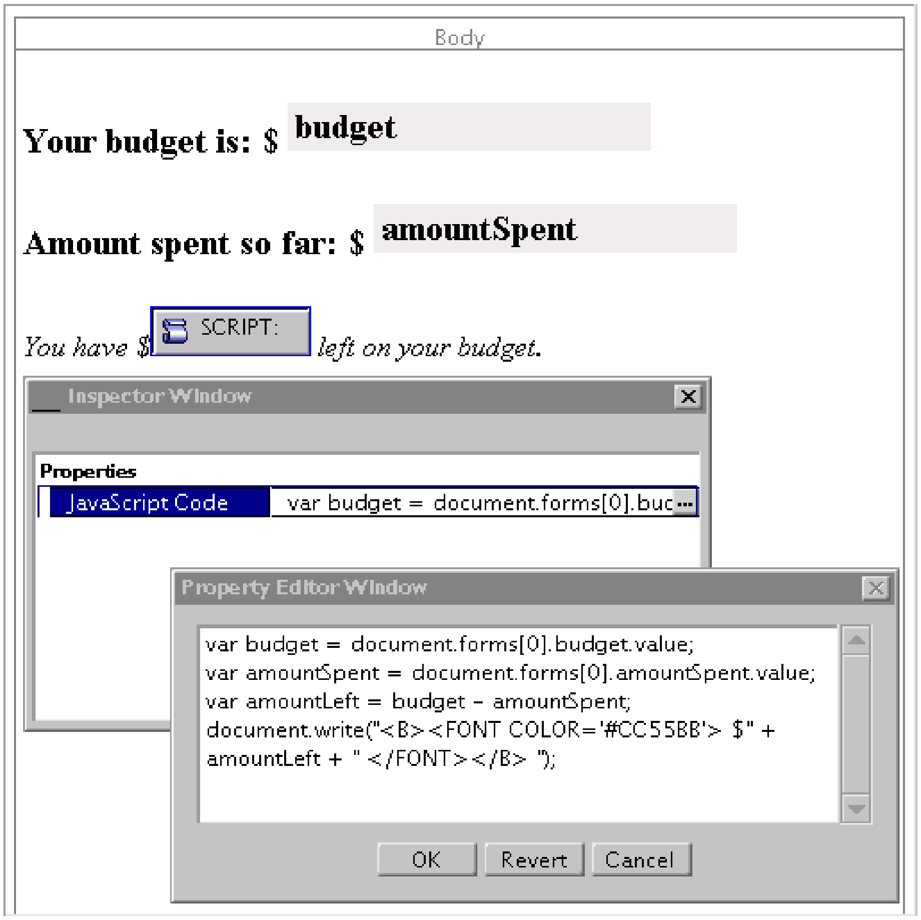
Thus, client-side scripts can access the value of any data field as it is currently displayed in the form. This is done by accessing the value of a form element that has the same name as the data field. Client-side scripts cannot get the value of a data field that is not represented in the form.

For example, the following embedded client-side script displays a message indicating how much money is left on the budget:

```
var budget = document.forms[0].budget.value;
var amountSpent = document.forms[0].amountSpent.value;
var amountLeft = budget - amountSpent;
document.write("<B><FONT COLOR='#CC55BB'> $" + amountLeft + "
</FONT></B> ");
```

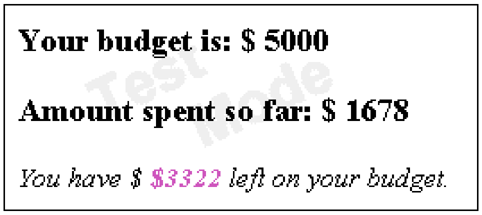
Figure 9-3 shows the definition of this embedded script in Process Builder, and Figure 9-4 shows the results of this embedded script in Process Express.

Figure 9-3 Defining an embedded client-side script in Process Builder



```
var budget = document.forms[0].budget.value;
var amountSpent = document.forms[0].amountSpent.value;
var amountLeft = budget - amountSpent;
document.write("<B><FONT COLOR='#CC55BB'> $" +
amountLeft + " </FONT></B> ");
```

Figure 9-4 Results of an embedded script in Process Express



Your budget is: $ 5000

Amount spent so far: $ 1678

You have $ $3322 left on your budget.

Note that an embedded script can access only those form elements that precede it physically in the page, because scripts are executed as the page is laid out. By contrast, event handlers can access other form elements no matter where they are in the page.

Chapter 10

Deploying an Application

This chapter describes how to deploy an application. Deployment is specified through the Deployment Dialog Box. After you deploy an application and test it, you'll typically need to change the application and redeploy it.

This chapter describes the following topics:

Before You Deploy

Deploying an application makes it available to test and use. However, before you deploy an application, make sure the following tasks have been performed:

- "Set up and Configure iPlanet Process Manager"
- "Deploy Subprocesses First"
- "Save the Process Map, If Desired"
- "Fix Application Errors"

Set up and Configure iPlanet Process Manager

Before you can deploy an application, you must have all the pieces of iPlanet Process Manager installed and configured, including the corporate user directory, the database, and Process Administrator. In addition, you must have a cluster set up to deploy it to. For more information on creating a cluster, see the *Process Administrator's and Business Manager's Guide*.

You must also have your cluster information included in your `preferences.ini` file. For more information, see "Editing the preferences.ini File" on page 54.

If you want to deploy your applications using https, see the section "Configuring Builder for Secure Deployment" on page 47.

If you want to deploy your applications through a proxy server, see the section, "Configuring Builder for Deployment Through a Proxy" on page 50.

Deploy Subprocesses First

If you are using subprocesses, the child process must be deployed before you use it in a parent process. In this way, all expected components are in place when you deploy the parent process.

Save the Process Map, If Desired

When an administrator is viewing the Work Items List produced by the Work Items Statistics page, it's convenient for the administrator to have an image of the process map. If you want to provide this image, you must save the process map as an image file before you deploy the application. To save the process map, choose "Save Process Map Image" from the Applications menu.

Fix Application Errors

Once you have built an application and filled in the necessary configuration information, you can begin deploying the application. During the deployment process, your application syntax is checked. If errors are generated, you must correct them before you can deploy the application. If only warnings are generated (and no errors are generated), you are allowed to deploy the application. However, doing so might lead to problems using the application.

Steps for Deploying an Application

➤ **To deploy an application**

1. With your application open, click Deploy, or open the Application menu and choose Deploy.

   If errors are reported, you must fix them first. If no errors are reported, the Deploy Application dialog box appears. For details about this dialog box, see "The Deploy Application Dialog Box" on page 208.

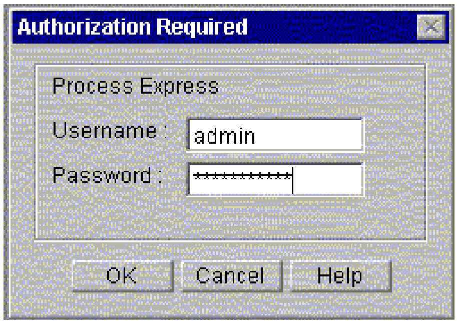
2. From the drop-down menu, select the cluster to which you want to deploy the application.

   If this is the first time you are deploying an application, the Authorization Required dialog box appears, prompting you to enter a Process Express username and password.

3. Enter your iPlanet Process Manager administration username and password.

   As an alternative, you can enter the username and password of someone with deployment privileges. By default, any administrator can deploy an application. A sample entry is shown in Figure 10-1:

Figure 10-1 The Authorization Required dialog box


Authorization Required

Process Express

Username : admin

Password : ************

4. Click OK to close the Authorization Required dialog box.

5. In the Deploy Application dialog box, the database user and database password fields are filled in automatically. Change them if necessary.

6. Select a stage for deployment. If you are still designing the application, you typically want to deploy it to Development. If the application is ready for people to use, deploy it to Production.

7. Choose whether the application is in testing mode or not.

   If testing mode is set to true, all work items in the application are assigned to the creator of the process instance. If set to false, the work items are assigned to the groups and user roles defined in the application.

8. Click OK.

   If your Web Server is configured for secure (https) deployment and the certificate sent by the Web Server is not stored in Process Manager's certificate store, a question dialog appears requesting your approval to accept a certificate. The dialog appears in Figure 10-2.

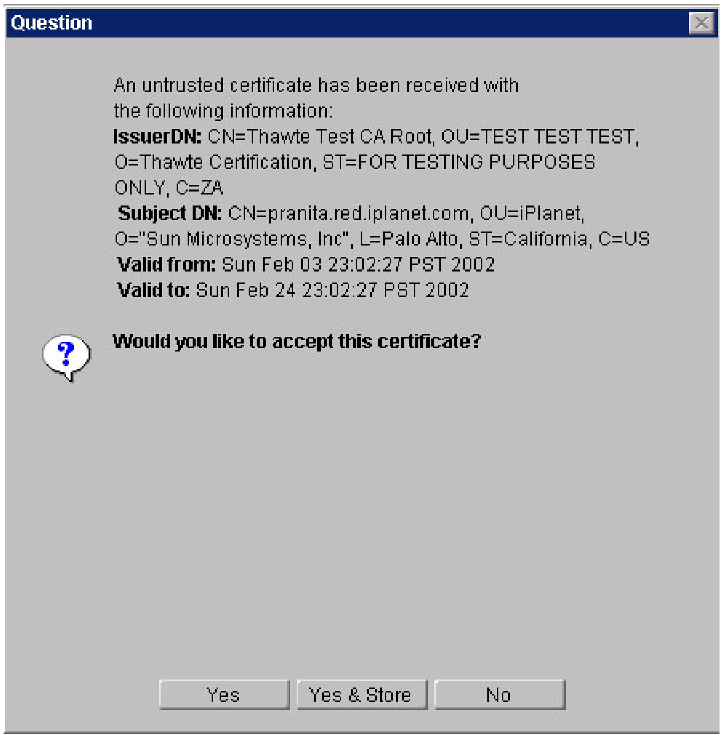
   For details about secure deployment using Process Builder, see "Configuring Builder for Secure Deployment" on page 47.

9. Accept or deny the certificate.

   You have three options.

   - Yes. If you choose Yes, Builder accepts, but does not store the certificate and the deployment continues.
   - Yes & Store. If you choose Yes & Store, Builder accepts and stores the certificate and the deployment continues.
   - No. If you choose No, Builder denies the certificate and the deployment terminates.

Figure 10-2 Certificate Acceptance Dialog

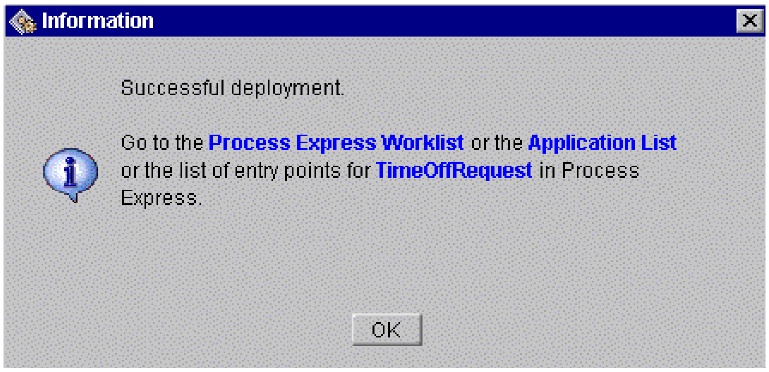


An untrusted certificate has been received with the following information:
IssuerDN: CN=Thawte Test CA Root, OU=TEST TEST TEST, O=Thawte Certification, ST=FOR TESTING PURPOSES ONLY, C=ZA
Subject DN: CN=pranita.red.iplanet.com, OU=iPlanet, O="Sun Microsystems, Inc", L=Palo Alto, ST=California, C=US
Valid from: Sun Feb 03 23:02:27 PST 2002
Valid to: Sun Feb 24 23:02:27 PST 2002

Would you like to accept this certificate?

If deployment succeeds, an Information dialog box appears. A sample is shown in Figure 10-3:

Figure 10-3 A successful deployment brings up the Information dialog box



Successful deployment.

Go to the **Process Express Worklist** or the **Application List** or the list of entry points for **TimeOffRequest** in Process Express.

10. From the Information dialog box, you have two main choices:

    - Click OK to close the dialog box and resume working in Process Builder.
    - Click one of the highlighted links (Process Express Worklist, Application List, or the application name) to launch Process Express at a particular location.

    Solaris Only. In order for the links to work on Solaris, you must add the following line in your preferences.ini file:

    browser = *netscape_root*/netscape -remote openURL{(0)}

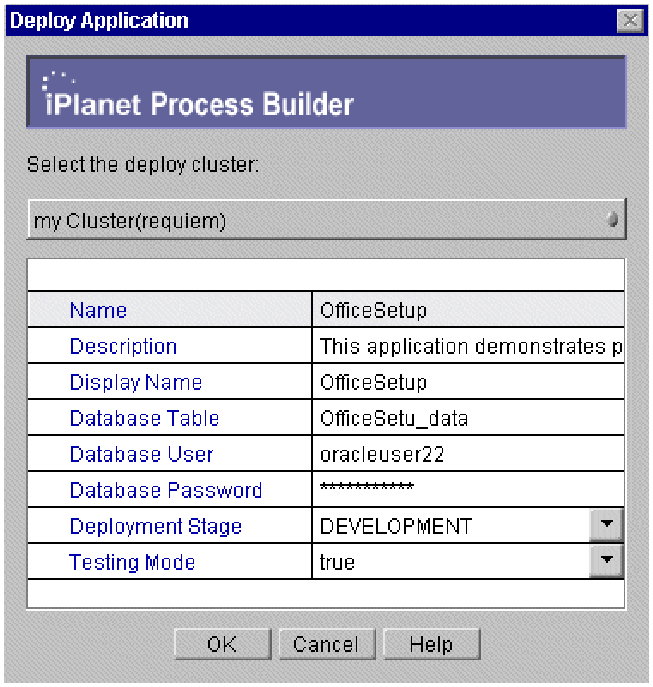
    where *netscape_root* is the directory in which your Netscape browser is installed. Without the previous line, your browser will not open when you click the links in the Information dialog box.

After you deploy the application, it is available for use.

The Deploy Application Dialog Box

Figure 10-4 shows the Deploy Application dialog box, where you enter deployment information.

Figure 10-4 The Deploy Application dialog box



Select the deploy cluster:

my Cluster(requiem)

Name	OfficeSetup
Description	This application demonstrates p
Display Name	OfficeSetup
Database Table	OfficeSetu_data
Database User	oracleuser22
Database Password	************
Deployment Stage	DEVELOPMENT
Testing Mode	true

Name The name of the application that appears in the list of applications. You cannot edit this field.

Description A longer description of the application. This description appears in Process Express. The user has to have enough information between the Display Name and the Description to identify the application they want from a list of applications, so it's important to make these two fields descriptive.

Display Name The longer, more descriptive name displayed by Process Express. If you are updating an existing application by saving it with a new application name, you can still keep the same Display Name, so that the name users see can be the same from version to version of the application.

Database Table The table in the database where the data for the application is stored. This table is created automatically when you deploy the application. The table name for each application must be unique. This field defaults to the application name. Because the table name can be no more than 15 characters long, if your application name is longer, it truncates the application name. This field is required.

Database User The user name for accessing the database where the application's data is stored. This field is required. This user can be your cluster database user, or you can use a more restrictive application-specific database user. The default is the cluster database user. If you use a a more restrictive application-specific database user, then the cluster database user must have privileges to create tables in the table space of the application-specific database user. Note: While this feature is more appropriate for Oracle database users, it may not be as useful for other database users. You must not change the user after the application is deployed. If you do, you may get errors when the application is run.

Database Password The password used to access the database. This field should be left unmodified. It is filled with the cluster database user's password. Changing this field has no affect on the application runtime.

Deployment Stage Indicates whether the application is in Development or in Production. The deployment stage governs what you can change in the application. In Development, you can change all application information. In Production, there are some application changes you are not allowed to make.

Testing Mode This property designates whether the application is in testing mode or not. If set to true, the application is in testing mode. In testing mode all work items are assigned to the process initiator. You can deploy in testing mode in either the Development or Production stage. This field is required.

Revising a Deployed Application

The deployment stage affects the types of revisions you can make to an application. You can deploy an application either to Development or to Production:

- Development

  When you deploy with the deployment stage set to Development, you can test the deployed application. After testing, you can change anything about the application. For example, you can deploy an application, test it, and decide to add or delete data fields.

  Because the application is still in development, iPlanet Process Manager assumes that you do not need to preserve data or open work items. As a result, if you make changes to the data dictionary by adding or deleting data fields, iPlanet Process Manager replaces the whole application when you redeploy it. This causes all existing process instances to be deleted.

  However, if you do not make changes to the data dictionary, your existing process instances will still be there if you redeploy the application.

- Production

  After you deploy an application with a deployment stage of Production, you can still change certain application information if you need to. For example, you can add new activities to an application that is deployed to Production.

Summary of Allowed Revisions

This section lists the different aspects of an application that you can change. The summary lists are grouped as follows:

- "Changes to Activities and Transitions"
- "Changes to Data Elements"
- "Changes to Forms, Scripts, and Content Store"

Changes to Activities and Transitions

Table 10-1 summarizes the revisions allowed to activities and transitions, depending on the application's deployment stage.

Table 10-1 Changes allowed to application's activities and transitions

Task	**Development Stage**	**Production Stage**
Add activity	Allowed	Allowed
Add transition	Allowed	Allowed
Remove activity	Allowed	Not allowed
Remove transition	Allowed	Allowed
Rename activity	Allowed	Not allowed
Rename transition	Allowed	Allowed
Remove entry or exit point	Allowed	Allowed

Changes to Data Elements

Table 10-2 summarizes the revisions allowed to data elements, depending on the application's deployment stage.

Table 10-2 Changes allowed to application's data elements

Task	Development Stage	Production Stage
Add data field	Allowed	Allowed
Remove data field	Allowed	Not allowed
Rename data field	Allowed	Not allowed
Change data field type	Allowed	Not allowed
Change data field size	Allowed	Not allowed
Change data field properties	Allowed	Allowed only on the following properties: Allow Search Default Value Display Name Description Help Message On Blur On Focus On Value Change

Changes to Forms, Scripts, and Content Store

Table 10-3 summarizes the changes allowed to forms, scripts, and the content store in deployed applications. These changes are allowed in both the development stage and in the production stage.

Table 10-3 Changes allowed in both development and production stages

Allowed form changes	Allowed script changes	Allowed content store changes
Add form	Add scripts	Change content store URL
Edit form	Edit scripts	Change content store authorization

Table 10-3 Changes allowed in both development and production stages *(Continued)*

Allowed form changes	Allowed script changes	Allowed content store changes
Delete form	Delete scripts	
Allow user access to form		
Remove user access to form		

Deployed Applications Compared with Local Copies

In most cases, you want to edit the deployed version of an application, not the version stored locally. You'll want to do this because the deployed version is usually the most current. If you edit the local version and then deploy it, you will overwrite your current deployed version.

NOTE If your local version has a unique application ID that differs from that of your deployed version, you will not be allowed to deploy the local version.

Using a Backup of a Local Application

When you open a deployed application, Process Builder saves a copy of the application locally, in the `builder/Applications/`*application_name* folder. But you may already have a version of the same application stored locally. In that case, opening the deployed version will launch a dialog box that asks whether you want to overwrite the local version with the deployed version.

If you choose to overwrite the local version, Process Builder saves a copy of the local version in a file called *application_name_backup*`.zip`. In this way, you'll have a backup of your local version in case you need to revert to it. For example, the deployed version might be corrupted, or it might not contain recent changes that you made to the local version. To revert to the local backup version, unzip the file.

Saving a Local Application to Another Name

Sometimes you want to make changes to an application, but the changes are not allowed at the deployment stage. In that case, use Save As to rename the application, and then make the changes in the new application.

➤ **To rename an application**

1. Open the most recent version of the application (usually the one deployed to the cluster).
2. From the Application menu, choose Save As.
3. In the dialog box that appears, rename the application.

   Note that you can use the same name if you save the application to a different local folder, but the new application will have a different unique ID. You will not be able to overwrite the deployed version of the application with your updated version, even though the name is the same, because the unique ID is different.
4. In the application tree view, double-click the application name to access the properties.
5. Make any changes required to the application, and redeploy under its new name.

Redeploying an Application

After you have deployed an application, you can change it and redeploy it.

- If the deployment stage is Development, redeploying deletes any open work items and data (if they are affected by the application revisions you made).
- If the deployment stage is Production, your open work items and data are preserved.

When you redeploy an application, the application's stage (shown in Process Administrator) is set to the stage of the deployed application, not the locally stored application. For example, if the application in the cluster had a stage of Open, and the local application had a stage of Testing, the redeployed application's stage is set to Open. For more information on application stages, see the *Process Administrator's and Business Manager's Guide*.

Deploying an Application to More Than One Cluster on the Same Machine

Each application you deploy to the application server receives an ID that is generated based on the name of the application. If you deploy an application on one cluster and then try to deploy the application using the same name on a different cluster, the application server does not allow the deployment because of the similar ID. If you want to deploy the same application to more than one cluster that resides on the same application server machine, you must rename the application before you deploy it.

For example, you want to deploy an application named ProcessOrder to a cluster named MyOracleCluster, and you want to deploy the same application to a cluster named MySybaseCluster. If you do not change the name of the application, the application server will not register the second instance of ProcessOrder.

NOTE The following procedure instructs you how to save applications in application directories you specify for each cluster. You do not have to set up your directory structure in this way. You can rename and save all of your application versions in Process Builder's default application folder.

➤ **To deploy an application to more than one cluster on the same application server**

1. Create application directories on your local machine, one for each cluster:
   - <*iPM_install_directory*>`/builder/MyOracleCluster/Applications`
   - <*iPM_install_directory*>`/builder/MySybaseCluster/Applications`
2. Open the `preferences.ini` file located in the following directory: <*iPM_install_directory*>\builder
3. Locate the `local_folder` entries in the `preferences.ini` file.
4. Add the following lines for your new local application directories, one for each cluster:

   ```
   local_folder = Oracle Cluster Apps;MyOracleCluster/Applications
   local_folder = Sybase Cluster Apps;MySybaseCluster/Applications
   ```

   The part of the assignment preceding the semi-colon is the name of the directory that appears in the builder UI when browsing for applications. The part of the assignment following the semi-colon is the path to the directory on the local machine.

You must provide the full path to the directory if it does not reside within the builder directory. Otherwise, you can provide the path relative to the builder directory as shown above.

5. Save the `preferences.ini` file.
6. Open the ProcessOrder application in the Process Builder.
7. Select the Save As command and save the ProcessOrder application into the `MyOracleCluster/Applications` directory using the application name `OraProcessOrder`.
8. Select Open and Browse to the original ProcessOrder application.
9. Select the Save As command and save the ProcessOrder application into the `MySybaseCluster/Applications` directory using the application name `SybProcessOrder`.

You can now open both applications and deploy them to your clusters.

Deploying Applications with the Command Line Deploy Tool

The command line deploy tool allows you to deploy one or more applications created in the Process Manager Builder. It is not a standalone substitute for Builder. The command line deploy tool delivers the deployment functionality of the Process Manager Builder deploy function without having to use the Builder's UI.

The command line deploy tool has three dependencies:

- A deployment descriptor (`.dd`) file. Inside this text file, you determine the deployment settings required to deploy/test your application.

  If you do not provide a deployment descriptor file, the command line deploy tool supplies the default values used in a typical development deployment from within Builder.

- A `PMExtensionManager.properties` file. This is the cluster boot file. It contains the address of the application's target cluster configuration directory. When you create a cluster, Process Manager creates the `PMExtensionManager.properties` file in the cluster's directory: `/iPlanet/ipm65/bpm/<`*ClusterName*`>/`. In a single cluster implementation, the default cluster name is `PMCluster`.

By default, the command line deploy tool looks in the local directory for the cluster boot file. You can move this file to any location, but you must reference its location at deploy time.

- An exported Process Manager Builder application `.zip` file.

  By default, the command line deploy tool looks for the application `.zip` file in the local directory. You can export the file to any location, but you must reference its location at deploy time.

NOTE The command line deploy tool does not check for individual application errors. Make sure to check each of your applications for errors in builder before deploying them with the command line deploy tool.

Deploying an Application with the pmdeploy Command

1. Export a Builder application to a zip file.
2. Create a deployment descriptor (.dd) file for the application.

   If you choose not to create a deployment descriptor file, the Command Line Deploy Tool supplies the default values used in a typical development deployment from within Builder.
3. Invoke the pmdeploy command from the command line.

   You can choose to deploy a single application from the command line or deploy multiple applications using a batch file or a makefile.

A Typical Deployment Descriptor .dd File

```
corpdirurl: URL to the corporate ldap directory
prettyname: Name of the application
description: Description of the application
dbuser: User name for the database
dbpassword: Password for the database
dbdatatablename: Name of the database table
appstage: DEVELOPMENT/PRODUCTION
apptesting: true/false
```

You can also add EJB references to the deployment descriptor file. To learn more about adding EJB references to an application's deployment descriptor file, see "Adding EJB References to an Application Using the Command Line Deploy Tool" on page 256.

Syntax for pmdeploy

```
pmdeploy -U <username> -P <password> -apppath <path_to_app.zip>
-deployfilepath <path_to_app.dd> -bootfilepath
<path_to_PMExtensionManager.properties> <appname>
```

The only *required* values in the pmdeploy command are the username, the password, and the appname.

NOTE The `pmdeploy` command does not consult Builder's `preferences.ini` file when deploying an application.

Other *optional* values that can be used in the pmdeploy command:

Table 10-4 pmdeploy Command Options

Optional Value	Description
`-apppath`	path to the application zip file By default, the command line deploy tool looks for the application's zip file in the local directory. You can export the file to any location, but you must reference its location at deploy time.
`-deployfilepath`	path to the .dd deployment descriptor file If you do not provide a deployment descriptor file, the command line deploy tool supplies the default values used in a typical testing deployment from within Builder.

Table 10-4 pmdeploy Command Options *(Continued)*

Optional Value	Description
`-bootfilepath`	path to the PMExtensionManager.properties file
	When you create a cluster, Process Manager creates the `PMExtensionManager.properties` file in the cluster's directory: `/iPlanet/ipm65/bpm/<`*ClusterName*`>/`. In a single cluster implementation, the default cluster name is `PMCluster`.
	By default, the command line deploy tool looks for the cluster boot file in the local directory. You can move this file to any location, but you must reference its location at deploy time.
`-proxyHost`	name of the proxy host when deploying an application through a proxy server
	You must supply a proxy host when deploying an application through a proxy server.
`-proxyPort`	number of the proxy port when deploying an application through a proxy server
	If you specify a proxy host and no proxy port, the command line deploy tool supplies a default proxy port of 80.
`-dontAskCert`	setting this option results in an exception and failed deployment if an unknown certificate is encountered during secure (https) deployment
`-forceAcceptCert`	setting this option results in automatic acceptance of an unknown certificate when it is encountered during secure (https) deployment
	This option takes precedence over `dontAskCert`.
`-h`	print usage help for the `pmdeploy` command
`-help`	print usage help for the `pmdeploy` command
`-verbose`	display debug data on the console/standard err
`-description`	description of the application being deployed

When the web server and Builder are configured for secure deployment, the pmdeploy command requires you to accept or decline any unknown certificates. To accept an unknown certificate, type "`y`" when prompted and the deployment proceeds. To decline an unknown certificate, type "`n`" when prompted and the deployment terminates. To override this prompt, use the options `dontAskCert` and `forceAcceptCert` as described in Table 10-4.

NOTE The pmdeploy command does not allow you to store an unknown certificate during deployment. You can only accept an unknown certificate. To store an unknown certificate, you must deploy the application using the graphical Builder tool's deploy command.

For details on configuring iPlanet Process Manager for secure deployment of applications, see the section, "Configuring Builder for Secure Deployment" on page 47.

Chapter 11

Advanced Techniques for Scripting

This chapter describes how to write your own scripts for use with Process Builder.

This chapter has the following sections:

Introduction

When writing a Process Manager script, consider the types of information your script will use. Process Manager scripts can use the following sources of information:

- Information about the process in progress.

  This data is stored in Process Manager's database, and is available through the `processInstance` object. This information includes data such as the date the process instance was created and the current value of fields such as the `specialDiscount` field.

- Information about the work item in progress.

  Information about the current work item is available through the `workItem` object. This information includes data such as the user to whom the work item is assigned.

- Information about users.

  This information is stored in the Directory Server, and is available through the `corporateDirectory` object, discussed in "Getting Information about Users and their Attributes." This data includes information about users, such as the assignee's address or the name of their manager.

- Items stored in the content store.

  Scripts can access items stored in the content store by using the `contentStore` object (discussed in "Accessing the Content Store" on page 226).

Getting Information about the Current Process

Process Manager provides two JavaScript objects, `processInstance` and `workItem`, that scripts can use to access information about the process in progress:

- `processInstance` holds information about the process instance in general, such as its creation date and the current values of any data fields. Scripts can use the global function `getProcessInstance()` to get the object. Scripts can then call methods on the `processInstance` to get and set data relevant to the process instance.

- `workItem` holds information about the current work item, such as its assignee and its expiration date. Scripts can use the global function `getWorkItem()` to get the object. Scripts can then call methods on the `workItem` object to get and set data relevant to the work item.

See Appendix A, "JavaScript API Reference" for details on these methods.

Getting and Setting Data Field Values

When a user submits a form for a work item, the value of each data field in the form is put into the corresponding data field in Process Manager's database. For example, if the user sets the value of the field `documentName` to `Jo Writer` then when the form is submitted the field `documentName` in Process Manager's database gets the value `"Jo Writer"`. (You could think of the field in the form as being a window to the field in the database.)

There are several ways to verify the data that users enter in the form fields, as discussed in "Verifying Form Input" on page 230.

Scripts can use the following methods on the `processInstance` object to get and set data field values:

- `getData (`*fieldName*`)`

  This method returns the value of the named field.

- `setData (`*fieldName, fieldValue*`)`

  This method sets the value of the named field to the given value. This method is particularly useful for setting values in automation scripts.

The following function is an example of a completion script that gets and sets field values:

```
function setOrderPrice () {
  // Get the process instance.
  var pi = getProcessInstance ();
  // Get the value of the item_price field.
  var price = pi.getData ("item_price");
  // Get the value of the number_ordered field.
  var numOrdered = pi.getData ("number_ordered");
  // Calculate the total price.
  var totalPrice = price * numOrdered;
  // Put the total price in the order_price field.
  pi.setData ("order_price", totalPrice);
  // Successful completion scripts return true.
  return true;
}
```

Getting Data Field Values in Decision Point and Automation Script Transitions

The value of a condition for a decision point or an automated script can be any expression that returns `true` or `false`.

These expressions can use a shorthand notation to refer to field values by simply using the name of the field. For example:

```
item_Price < 100
```

is shorthand for

```
getProcessInstance().getData("item_price") < 100
```

This expression evaluates to `true` if the value of the `item_price` field is less than 100, otherwise it evaluates to `false`.

Note that only scripts used in conditions can use this shorthand for accessing data fields.

Getting Information about Users and their Attributes

Scripts can use the global function `getCorporateDirectory()` to get an object that represents the corporate directory in the Directory Server that Process Manager uses.

The Directory Server contains a database of users and their attributes, such as their distinguished name, their common name, their phone number, their address, their email address, and so on. Given the `corporateDirectory` object, scripts can get information about the users in the corporate directory.

Finding Users and Accessing their Attributes

The `corporateDirectory` object has three methods that allow scripts to search for users in the corporate directory:

- `getUserByCN (`*userCN*`)`
- `getUserByDN (`*userDN*`)`
- `getUserById (`*userID*`)`

These methods all return a JavaScript object that represents the user if they can be found; otherwise the methods return `null`. The object has variables for all the user's attributes. See Appendix A, "JavaScript API Reference" for details.

Scripts can also use the following two methods on the `processInstance` object to get user attributes from the Directory Server:

- `getCreatorUser()` returns a JavaScript object containing the attributes of the user that created the process instance.
- `getRoleUser(roleName)` returns a JavaScript object containing the attributes of the user assigned to the given role.

For example, the following assignment script assigns the work item to the user whose user id is the value of the `peerReviewerID` attribute of the person who created the process instance.

```
assignToPeerReviewer () {
  var pi = getProcessInstance ();
  var creator = pi.getCreatorUser ();
  // Assignment scripts return an array of distinguished names.
  return new Array (creator.peerReviewerID);
}
```

Note that this script works only if the `peerReviewerID` attribute is defined in the Directory Server and is initialized for the creator user.

Modifying User Attributes

The `corporateDirectory` object has the following methods for adding, replacing, or deleting user attribute values in the Directory Server:

- `modifyUserById (`*userID*`, `*attrName*`, `*attrValue*`, `*operation*`)`
- `modifyUserByCN (`*userCN*`, `*attrName*`, `*attrValue*`, `*operation*`)`
- `modifyUserByDN (`*userDN*`, `*attrName*`, `*attrValue*`, `*operation*`)`

These methods specify the user either by ID or common name as appropriate. The `attrName` parameter is the name of the attribute to be modified. If `operation` is `ADD`, the value `attrValue` is added to any existing values. If operation is `REPLACE`, all existing values are replace by `attrValue`. If operation is `DELETE`, then `attrValue` is deleted from the attribute. The functions return `true` if the operation is successful, otherwise they return `false`.

For example, the following code puts the value `nikki@company.com` to the `mail` attribute of the user whose common name is `Nikki Beckwell`:

```
var corpdir = getCorporateDirectory();

corpdir.modifyUserByCN ("Nikki Beckwell", "mail",
"nikki@company.com", "REPLACE");
```

Although scripts can modify attribute values, they cannot create attributes that have not already been defined in the Directory Server schema.

Adding and Deleting Users

The `corporateDirectory` object has the following methods for adding and deleting users to the corporate directory:

- `deleteUserByCN (`*userCN*`)`
- `deleteUserByDN (`*userDN*`)`
- `deleteUserById (`*userID*`)`
- `addUser (`*userDN*`,` *attributes*`,` *objectClasses*`)`

See Appendix A, "JavaScript API Reference" for details.

Accessing the Content Store

Scripts can use the global function `getContentStore()` to get a `contentStore` connected to the content store for the process.

The following global function is useful for constructing a file name when you want to add an item to the object store:

- `getBaseForFileName()` returns the folder name that the current process instance uses to contain its stored content.

For more information about methods on the content store, see Appendix A, "JavaScript API Reference."

For details about configuring the content store for use with a secure web server, see "Enabling Content Store Over https" on page 48.

Example of Accessing a Stored Item

For an example of accessing the content store, consider a process that writers use for submitting documents for review. Before starting the process, the writer must create a file containing a synopsis of the document.

The entry point form contains a File data field named `docDescription`. The writer selects a file containing the document synopsis and uploads it by pressing the File icon in the form. The next activity after the entry point is an automated activity that uses the following automation script. This script gets the document synopsis by getting the content of the file that the writer selected. The script then puts the synopsis into the `docSynopsis` TextArea datafield. The intention here is that the next form in the process displays the synopsis to the users assigned to review the document.

```
function getDocDescription (){
  // Use the getProcessInstance() global function to get
  // a reference
  // to the process instance.
  var pi = getProcessInstance();

  // Retrieve the URL where the file containing the document
  // description is stored in the content store. This was set
  // when the writer selected a File containing a document
  // description.
  var descriptionURL = pi.getData("docDescription");

  // Get a reference to the content store service through the
  // global function getContentStore().
  var contentStore = getContentStore();

  // Get the content of the file.
  var docDescription = contentStore.getContent(descriptionURL);

  // Put the document description in a text field
  // called docSynopsis.
  pi.setData ("docSynopsis", docDescription);
  // Return true to proceed to the next activty.
  return true;
}
```

NOTE This script requires that the content store URL defined for the application is a valid URL, such as:

`http://pm.company.com/pmstore`

If the content store URL is not defined correctly, the `store()` method will not work.

Storing Files in the Content Store

Each Process Engine uses a series of directories named `part0`, `part1`, `part2` and so on to contain items in the object store. Each process instance has a unique identity, such as `pi0001`. Each process instance uses a `part`*n* subdirectory to store all its content. For example, all the stored files associated with process instance `pi0001` would be stored in *PM_root*`/part0/pi0001`. When `part0` gets full, the Process Engine starts using `part1`, and so on.

When a script needs to save content to the content store, it can use the `getBaseForFileName()` function to get the pathname for the content store directory for the current process instance. The script can then use the `store()` method on the `contentStore` object to save content to a file in the correct content store directory for the current process instance.

For example, the following automation script code creates a path name for the file `myNewFile.html` in the content store, then saves some content to a new file in the content store.

For details about configuring the content store for use with a secure web server, see "Enabling Content Store Over https" on page 48.

```
function storeSomething () {
  var pi = getProcessInstance();
  // Create a pathname for a new file in the content store.
  var pi = getProcessInstance ();
  var pid = pi.getInstanceId();
  var contentStorePath = getBaseForFileName (pid);
  var newFileName = contentStorePath + "myNewFile.html";

  // Create a string containing some content to be saved to the
file.
  var newContent = "Do something that generates a content string
here";

  // Store the content in the file myNewFile.html.
  var myStore = getContentStore ();
  myStore.store (newContent, newFileName);
  // Return true to proceed to the next work item
  return true;
}
```

Logging Error and Informational Messages

Process Manager provides several global functions for providing messages. Scripts can use these functions to log error messages, to add entries to the history log, and to add informational messages for the user when a script fails.

These functions are:

- `logErrorMsg()`
- `logInfoMsg()`
- `logSecurityMsg()`
- `logHistoryMsg()`

For more information about these global functions, see Appendix A, "JavaScript API Reference."

Verifying Form Input

This section has the following subsection:

- "Verifying Form Input with Client-Side JavaScript"
- "Verifying Form Data in Completion Scripts"

When you create a form in Process Builder, you place data fields on it. If the data field is in edit mode, it displays the current value, if any, and allows the user to modify the value. If a data field is in view mode, it displays the value of the field but does not allow the user to change it.

When the user submits the form by pressing an action button, the values in the data fields in the form are put in the fields in the process instance's table in the database. By default, Process Manager verifies that the data entered in each field is the correct type, and if there are obvious errors, it rejects the work item or entry point and displays a warning dialog box. For example, if the user enters the string value "`whenever`" into a data field whose type is `Date`, Process Manager catches the error and rejects the work item. (Note however that the `Date` verification is not very strict, if the user enters 6666/7777/8 as a `Date`, Process Manager accepts it.)

There will be times when your form needs to perform additional data verification, such as checking that a number falls within a certain range of numbers. Process Manager provides the following way to verify that users enter valid data for a field:

- use client-side JavaScript scripts to define `onClick` or `onValueChange` properties for those data fields that have them, or to define `onSubmitForm` scripts.
- use `onCompletion` scripts to perform data type verification.

Using client-side scripts to check data field values allows you to write scripts that give the user a chance to fix the values before the form is submitted.

If the data verification occurs in a completion script, then if the script fails the result is that the form disappears and the process is routed to an exception node.

Verifying Form Input with Client-Side JavaScript

Process Builder provides two ways to write client-side scripts to verify form input:

- using an embedded script that defines an `onSubmitForm` function that gets invoked when the form is submitted.
- using `onValueChange` and `onClick` event handlers for specific form elements.

For every data field represented in a form, the form contains a form element that has exactly the same name as the data field. Client-side scripts can access data field values as they are currently displayed in the form. The scripts do this by accessing form elements that have the same name as the data field.

If the form contains an embedded client-side script that defines an `onSubmitForm` function, that function is invoked when the user presses an action button on the form. You can define an `onSubmitForm` function to check values of form elements and offer the user a chance to enter new values in cases where the current values are invalid. If the `onSubmitForm` function returns `false`, the form is not submitted. See the section "onSubmitForm Example" on page 233 for a coded example of an `onSubmitForm` function.

You can also define client-side scripts to check the value of individual form elements. Some data fields have an `onClick` or `onValueChange` property. The value of this property is a JavaScript expression. These properties correspond to event handlers in the associated form elements. For more information about form elements, refer to any reference on HTML tags.

The `onClick` script is fired when the form element is clicked. For elements whose value is changed by clicking, such as radio buttons, the `onValueChange` script behaves just like an `onClick` script, and is fired whenever the user clicks the form element. For other elements, such as text fields, the `onValueChange` script is fired when the form element loses focus after its value has changed, which usually happens when the user presses the return key or clicks elsewhere in the form. To see exactly which data fields have `onClick` and `onValueChange` scripts, see the properties for the different kinds of data fields in Process Builder.

You can define `onClick`, `onValueChange`, and `onSubmitForm` scripts that display an alert box or a confirm box to warn the user that an invalid value has been entered. Use the `alert()` function to display an alert box, and use the `confirm()` function to display a confirm box.

To obtain information about JavaScript errors, enter the URL `javascript:` in your version of Netscape Communicator.

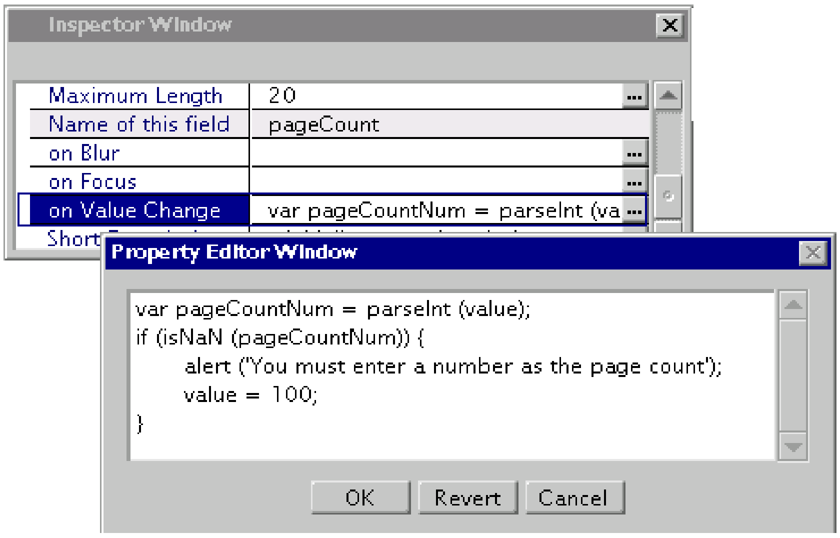
Event Handler Example

The following `onValueChange` script for a text field checks if the value of the text field is a number. If the value is not a number, it displays a dialog box with a warning and resets the page count value to 100 in case the user ignores the warning. If the value is a number, it does nothing. (Note that in event handlers, such as in client-side scripts, you must use single quotes instead of double quotes.)

```
var pageCountNum = parseInt (value);
if (isNaN (pageCountNum)) {
  alert (’You must enter a number as the page count’);
  value = 100;
}
```

Figure 11-1 shows the definition of this `onValueChange` event handler in the Inspector window in Process Builder.

Figure 11-1 A definition for an onValueChange event handler

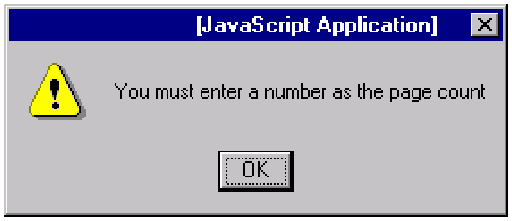


Maximum Length	20
Name of this field	pageCount
on Blur	
on Focus	
on Value Change	var pageCountNum = parseInt (va
Short	

```
var pageCountNum = parseInt (value);
if (isNaN (pageCountNum)) {
      alert ('You must enter a number as the page count');
      value = 100;
}
```

Figure 11-2 shows a sample alert window in Process Express:

Figure 11-2 A sample alert window in Process Express



You must enter a number as the page count

This script can be used directly as the value of the `onValueChange` property of the text field. The script uses `parseInt()` to convert the value of the text field from a string to a number. The `value` argument to the `parseInt` function is the name of a property of the text field. Since this script is used directly in the text field object, there is no need to say `this.value`, although the script would also work if written as:

```
var pageCountNum = parseInt (this.value);
// etc...
```

For information about the `parseInt()` method, see Chapter 13, Global Functions in the JavaScript reference at:

```
http://developer.netscape.com/docs/manuals/communicator/jsref/conte
nts.htm
```

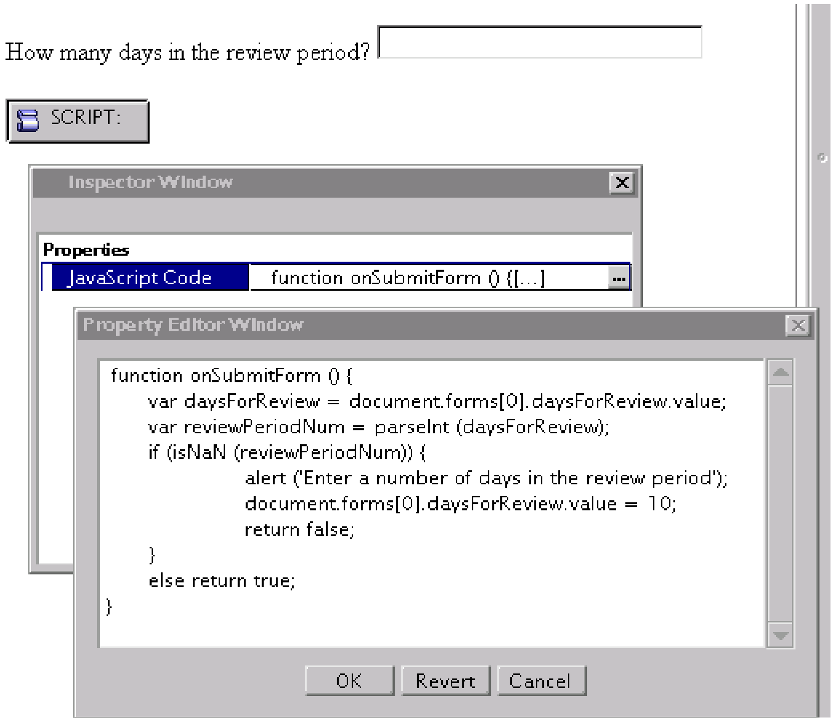
onSubmitForm Example

The following example shows an embedded client-side script that defines an `onSubmitForm` function. In this case, the script checks that the value entered in the `daysForReview` form element is a number. If not, the script displays a warning dialog box, sets the value of the `daysForReview` form element to 10, and returns `false` to give the user a chance to change the value and submit the form again.

```
function onSubmitForm () {
  var daysForReview = document.forms[0].daysForReview.value;
  var reviewPeriodNum = parseInt (daysForReview);
  if (isNaN (reviewPeriodNum)) {
      alert ('Enter a number of days in the review period');
      document.forms[0].daysForReview.value = 10;
      return false;
  }
  else return true;
}
```

Figure 11-3 shows an embedded client-side script that defines an `onSubmitForm` function in Process Builder:

Figure 11-3 A script that defines an `onSubmitForm` function



```
function onSubmitForm () {
    var daysForReview = document.forms[0].daysForReview.value;
    var reviewPeriodNum = parseInt (daysForReview);
    if (isNaN (reviewPeriodNum)) {
            alert ('Enter a number of days in the review period');
            document.forms[0].daysForReview.value = 10;
            return false;
    }
    else return true;
}
```

Verifying Form Data in Completion Scripts

When the user presses an action button to submit a form, the values of the data fields on the form are installed into the global `processInstance` object. When the completion script runs, it can access data field values by using the `getData()` method of the global `processInstance` object. However, if the completion script fails, the newly installed values are removed from the process instance, the previous values are re-installed and the process is routed to an exception node.

Initializing and Shutting Down Applications

When defining a process definition, you can write initialisation and shutdown scripts for the application. The initialisation script is invoked when the *application* first starts, and the shutdown script is invoked when the application is shut down. Note that the initialization and shutdown scripts are not invoked each time a process instance starts or finishes, but only when the application starts or finishes. The application lifespan (start to finish) is defined within the scope of each Java engine running within the application server. Therefore, the initialization scripts are invoked on a per-engine basis.

Initialization scripts and shutdown script are not associated with individual process instances, therefore they cannot use `processInstance` or `workItem` objects. They can however use the methods on the `contentStore` object. For example, the initialisation script could use the `store()` method on the `contentStore` object to create a file that can be accessed by all process instances in the application.

You can use the initialization script to perform tasks that need to be done once in the lifetime of the application, such as creating files or initiating connections to be shared by all process instances.

The shutdown script should close any connections that were opened in the initialization script. Other uses for a shutdown script include cleaning up external database tables, sending email to the administrator, releasing other resources, and performing any tasks that need to be done when the application is shut down.

Debugging Hints

To check for syntax errors in your scripts, enter `javascript:` as your URL. In addition, this section provides hints on debugging and troubleshooting scripts. It has the following subsections:

- "Displaying the Progress of a Script"
- "Testing Expiration Setter and Handler Scripts"

Displaying the Progress of a Script

There are several global functions for printing messages that are very useful for debugging scripts. The `logErrorMsg()` function adds an entry to the error log. The `logInfoMsg()` function adds an entry to the informational log. The `logSecurityMsg()` function adds an entry to the security log.

To view these logs, go to the Process Administrator page (Administrator.apm). For example, if Process Manager is installed at `pm.company.com`, go to:

`pm.company.com/Administrator.apm/`

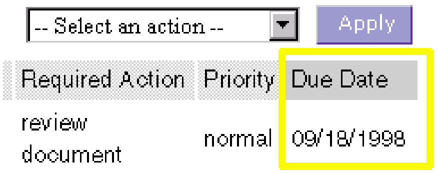
This page displays Process Administrator. Select the Applications tab (if it is not already selected) to see the list of applications. Select View Logs from the pop up menu for the application of interest, then press the Apply button. You may need to enter your user name and password to gain access to Process Manager and the Enterprise Server.

When the View Logs Server Selector window appears, select the desired cluster, then press the View Log File button. In the View Application Logs window that appears, you can select the Error, Security, or Information log. After making your selection, press the View Log button.

Testing Expiration Setter and Handler Scripts

When Process Express displays work items in the work item list, it shows the expiration date for each work item. To test an expiration setter script, simply deploy the application and test it out. Check the expiration date for the relevant work item in the work item list in Process Express. For example, Figure 11-4 shows the due date in Process Express.

Figure 11-4 Testing a due date



Required Action	Priority	Due Date
review document	normal	09/18/1998

At regular intervals (about once every minute) Process Manager tests for expired work items by comparing the expiration date and time of every work item to the current time. If the expiration time has passed for any work item, Process Manager invokes the work item's expiration handler script if it has one.

If your expiration handler re-assigns the work item, you should view the details and history for the process instance to check if the re-assignment worked in test mode. The title of the process does not change in test mode even if the assignee changes.

Sample Scripts

This section gives an example of the following kinds of scripts:

- "Assignment Script"
- "Expiration Setter Script"
- "Expiration Handler Script"
- "Completion Script"
- "Automation Script"

Assignment Script

The following assignment script assumes that the process instance has data fields `reviewer1` and `reviewer2` that contain the user IDs for two reviewers. If one or both of the `reviewer1` and `reviewer2` fields contain user IDs for valid users, the work item is assigned to the valid named reviewers. Otherwise it is assigned to the process instance creator.

The default value for the `reviewer1` and `reviewer2` fields is an empty string. The script checks that the reviewers exist as users in the corporate directory, and if so, assigns the work item to them. If the array does not contain valid user IDS, the function assigns the work item to the creator of the process instance.

```
function assignToNamedReviewers ()
{
  // Get the process instance and corporate directory
  var pi = getProcessInstance ();
  var corp = getCorporateDirectory ();


  // Create an array to contain the assignees.
  var assigneeArray = new Array ();


  // Get the user ids for reviewer1 and reviewer2.
  // Default values of revewer1 and reviewer2 data fields
  // are empty strings
  var reviewer1 = pi.getData ("reviewer1");
  var reviewer2 = pi.getData ("reviewer2");


  // Check that reviewer1 exists in the corporate directory
  if (reviewer1 != "") {
      reviewer1 = corp.getUserByID (reviewer1);
      if (reviewer1 != null) {
        assigneeArray.push (reviewer1);
      }
  }
  // Do the same thing for reviewer2.
  if (reviewer2 != "") {
      reviewer2 = corp.getUserByID (reviewer2);
      if (reviewer2 != null) {
        assigneeArray.push (reviewer2);
      }
  }



  // If the assignee array is not empty
  // return the assignee array otherwise return an array
  // of the creator of the process instance.

  if (assigneeArray.length != 0) {
      return assigneeArray;
  }
  else {
      return new Array (pi.getCreatorUser());
  }
}
```

Expiration Setter Script

This expiration setter script gets the value of the `pageCount` field and uses it to determine whether reviewers need a long or short review period. If the page count is greater than 100, the review period is long (14 days), otherwise the page count is short (7 days).

```
function setEndOfReviewPeriod(){
  // Get the process instance.
  var pi = getProcessInstance();

  // Get the page count.
  var pageCount = pi.getData ("pageCount");

  // The review period is based on the page count.
  // For pages < 100, expire in 7 days.
  // For pages > 100, expire in 14 days.
  var reviewPeriod;
  if (pageCount > 100) {
   reviewPeriod = 14;
  }
  else {
   reviewPeriod = 7;}

  var now = new Date();
  var nowTime = now.getTime(); //UNIX epoch time
  var expTime = nowTime + (reviewPeriod * 24 * 60 * 60 * 1000);
  return new Date(expTime);
}
```

Expiration Handler Script

This expiration handler script reassigns the work item to the creator.

```
function reviewPeriodExpired(){

  // Get the process instance and the work item.
  var pi = getProcessInstance ();
  var wi = getWorkItem();

  // Re-assign the work item to the creator
  wi.removeAssignees();
  wi.addUserAssignee(pi.getCreatorUser());

  return true;
}
```

Completion Script

The following completion script checks that the value of the `pageCount` data field is a number between 1 and 3000 inclusive. If the `pageCount` is too high, too low, or is not a number, the completion script logs an error message and returns `false`. The process instance is then routed to the appropriate exception node.

```
function checkPageCount(){
  var pi = getProcessInstance ();
  var pageCount = pi.getData("pageCount");

  // The standard JavaScript function parseInt gets an integer
  // from a string if it has one, otherwise returns 0.
  // For example, parseInt("33hello") returns 33.
  // If the first character cannot be converted to a number,
  // parseInt returns "NaN".
  var pageCountNum = parseInt(pageCount);

  // The standard JavaScript function isNaN
  // tests if a value is not a number.
  if ((isNaN (pageCountNum)) ||
     (pageCountNum < 1) || (pageCountNum > 3000)) {
      var e = new Object();
      e.exceptionString = "The page count was" + pageCount +
        " This value is invalid. " +
        "The page count must be a number " +
        "between 1 and 3000 inclusive.";
      logErrorMsg ("INVALID_PAGE_COUNT_ERROR", e);
      };
      return false;
  }
  else return true;
}
```

Automation Script

This script updates a link to a newly submitted document in a file that lists all documents available for review.

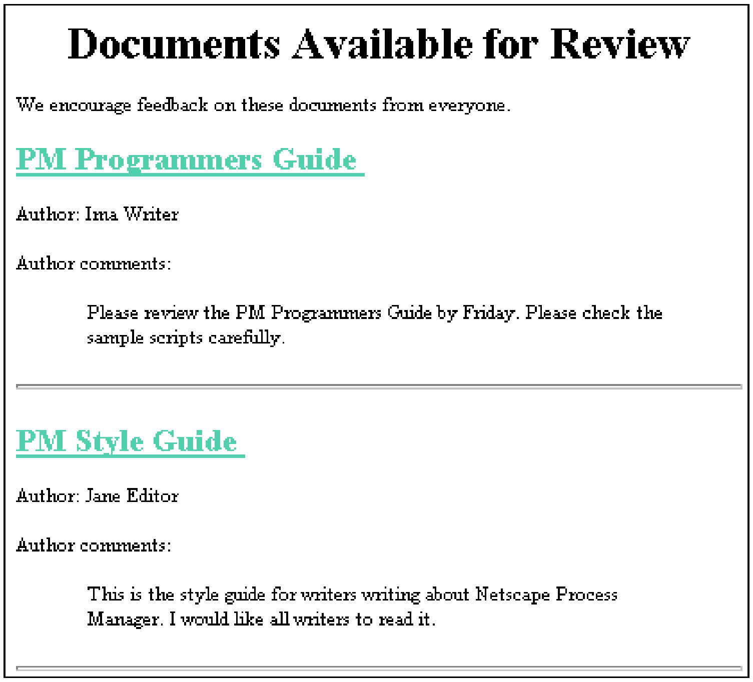
This script could be used as an automation script in a process that enables writers to submit documents for review. In the entry point form, the writer uses a File attachment data field called `docFileName` to attach the document. Many different writers can use the application to submit documents. The file `docList.html` contains an active link for every document that has been submitted with this application. The file `docList.html` lives in the top level of the content store so that all process instances in the application can access it.

Two files are involved in the document list. The file `docList.html` is an HTML-formatted list of the documents with an active link for each document. The file `showList.html` contains opening HTML tags, the list of documents (which is the same as `docList.html`), and closing HTML tags. This script uses two files so that it can add the closing HTML tags after adding the information for the newly submitted document.

```
function updateDocReviewWebPage(){
// Get the process instance.
var pi = getProcessInstance();

// docList.html and showList.html are stored at the
// top level in the content store.

// Get the root URL for the content store.
var myStore = getContentStore();
var storePath = myStore.getRootURL();

// Get the full pathnames to docList.html and showList.html.
var docListPath = storePath + "docList.html";
var showListPath = storePath + "showList.html";
// Get the contents of the doc list.
var docList = myStore.getContent(docListPath);

// Get the pathname to the doc that was submitted for review.
var docURL = pi.getData("docFileName");

// Get info about the newly submitted document.
var docname = pi.getData("docname");
var writer = pi.getData("writername");
var writerComments = pi.getData("writerComment");

// Add a link to the newly submitted doc in the doc list.
docList += "<A HREF=" + docURL + ">";
docList += "<H2><FONT COLOR='#55CCAA'>" + docname +"
</H2></FONT></A>";
docList += "<P>Author: " + writer + "</P>";
docList += "<P>Author comments: </P>";
docList += "<BLOCKQUOTE>" + writerComments + "</BLOCKQUOTE><HR>";

// Store the doclist back into docList.html
myStore.store (docList, docListPath);

// Create the final content, which contains the heading tags,
// the doc list, and the closing tags.
var finalContent = "<HTML><HEAD><TITLE>" +
  "Documents available for review </TITLE></HEAD>";

finalContent += "<BODY><CENTER><H1>" +
  "Documents Available for Review" +
  "</H1></CENTER><P>" +
  "We encourage feedback on these docs from everyone.</P>" +
  docList + "</BODY></HTML>";

myStore.store (finalContent, showListPath);

return true;
}
```

Figure 11-5 illustrates the `showList.html` web page. This page is updated each time a document is submitted for review.

Figure 11-5 A web page that is updated automatically



Documents Available for Review

We encourage feedback on these documents from everyone.

PM Programmers Guide

Author: Ima Writer

Author comments:

Please review the PM Programmers Guide by Friday. Please check the sample scripts carefully.

PM Style Guide

Author: Jane Editor

Author comments:

This is the style guide for writers writing about Netscape Process Manager. I would like all writers to read it.

Chapter 12

Scripting with EJB Components

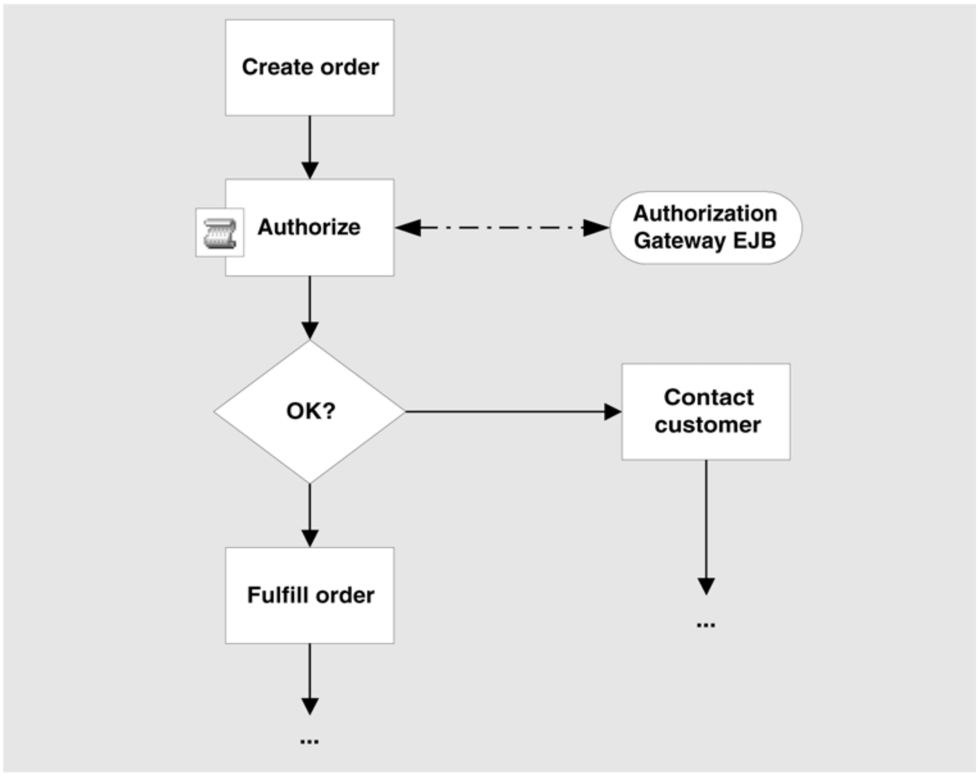
This chapter describes how to call an Enterprise JavaBean (EJB) from JavaScript. You should be familiar with EJBs to understand the information in this chapter.

This chapter describes the following topics:

Calling EJB Components from JavaScript

Your JavaScript process automation script can create an instance of an EJB and call the EJB's methods to perform business logic. For example, an order-entry application might create an instance of an EJB that represents a credit card authorization gateway. The application would call methods to check the credit card number, authorize the transaction, and create a charge on the account, as shown in Figure 12-1:

Figure 12-1 Programming logic for calling an EJB component



A Sample Script

To use an EJB, the calling JavaScript must perform the following actions:

- Look up the bean using the home interface's JNDI name
- Create an instance of the bean
- Call the bean's methods to perform tasks supported by the bean

The following JavaScript function shows an example of these actions:

```
function billCC ()
{
  var pi = getProcessInstance ();
  var wi = getWorkItem();
```

```
    pi.setData("ccError",null);
    var ccNumber = pi.getData("ccNumber");

    // Get bean home, create it, and call the check method on it.
    var home = ejbLookup("iPlanet/CreditCardServer");
    var ccs  = home.create ( pi.getData("ccType") );

    var result = ccs.check ( ccNumber );
    if (! result.equals
        ( Packages.com.netscape.pm.sample.ICreditCardServer.CARD_OK ))
    {
        pi.setData("ccError", result);
        return true;
    }

    result = ccs.authorize ( ccNumber, 100, 25 );
    if (result.startsWith (
        Packages.com.netscape.pm.sample.ICreditCardServer.CREDIT_LIMIT_EXCEEDED)
        || result.startsWith (
        Packages.com.netscape.pm.sample.ICreditCardServer.CREDIT_DENIED) )
    {
        pi.setData("ccError", result);
        return true;
    }

    // Charge the card the amount shown.
    result = ccs.charge ( ccNumber, result );
    if (! result.startsWith (
        Packages.com.netscape.pm.sample.ICreditCardServer.OPERATION_OK ))
    {
        pi.setData("ccError", result);
        return true;
    }

    pi.setData("ccAuthcode", result);
    return true;
}
```

You call the `ejbLookup()` function to create a reference to the EJB. This method uses the bean's JNDI name to determine the EJB to use. In this example, the name is `iPlanet/CreditCardServer`. You can examine the bean's `BeanHomeName` property to determine the JNDI name.

You call the bean's factory method, which is defined in the bean's home interface class, to create the instance of the bean. In this example, the method is `create()`. You can examine the bean's `HomeInterfaceClassName` property to determine the home interface class.

You can then call the bean's methods, which are defined in the bean's remote interface class, to perform specific tasks. In this example, the JavaScript calls the bean's `check()`, `authorize()`, and `charge()` methods. You can examine the bean's `RemoteInterfaceClassName` property to determine the remote interface class.

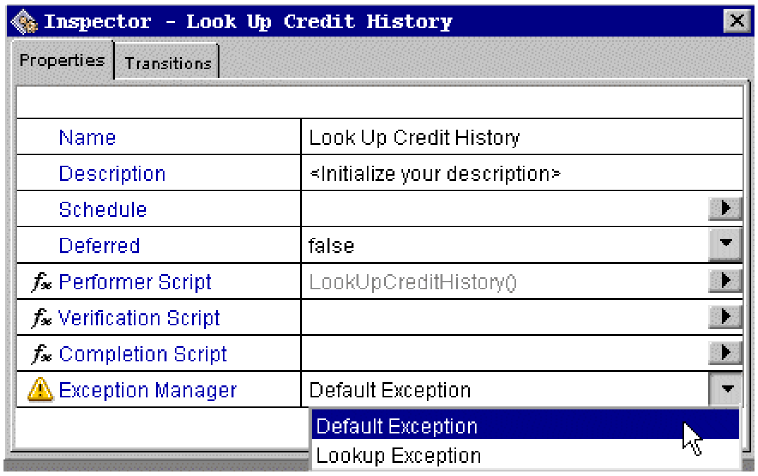
Handling Exceptions

Java exceptions cannot be handled in JavaScript. You can implement your bean so that the script receives error information without needing to deal with an exception, as in the following example:

```
...
catch (...Exception e) {
  e.PrintStackTrace();
     return true;
  }
}
```

If an exception is thrown, the process instance moves to the exception manager that you specify in the Inspector window, as shown in Figure 12-2:

Figure 12-2 Setting an exception manager in Process Builder



Name	Look Up Credit History
Description	<Initialize your description>
Schedule	
Deferred	false
Performer Script	LookUpCreditHistory()
Verification Script	
Completion Script	
Exception Manager	Default Exception

Default Exception
Lookup Exception

Data Conversion Issues

Process Manager uses LiveConnect's rules to convert between Java and JavaScript data types. When you call a Java method from JavaScript, JavaScript does not convert the data type of the data until it is manipulated by the script. For example, the `result` variable in the script contains a Java string data type after executing the Java EJB `check()` method in the following statement:

```
var result = ccs.check ( ccNumber );
```

You must compare the variable's contents using Java methods; comparing with JavaScript functions will not work as expected. The following statement shows the comparison using Java:

```
if (result.equals
( Packages.com.netscape.pm.sample.ICreditCardServer.CARD_OK )
  == false) { ... }
```

To access Java data, you must qualify the package name with the `Packages` keyword unless the Java data is from the `java` package. JavaScript recognizes the `java` package without qualification. You can also create an instance of an object by specifying the `new` operator, as in the following examples:

```
var aResult = new java.lang.String;
var CARD_OK = new
  Packages.com.netscape.pm.sample.ICreditCardServer.CARD_OK
```

NOTE Conversion between Java and JavaScript is implicit and occurs when the value is manipulated as a JavaScript data type. For example, if `result` contains a Java string data type, the following statement causes it to be converted to a JavaScript string data type:

```
result = result + "";
```

Adding EJB References to a Process Manager Application

In a J2EE environment, an EJB in one J2EE module or application that refers to another EJB in a different module or application must contain information about the EJB to which it refers. This referencing occurs in the deployment descriptor (`ejb-jar.xml`) file. You use an `<ejb-ref>` tag to declare the reference information.

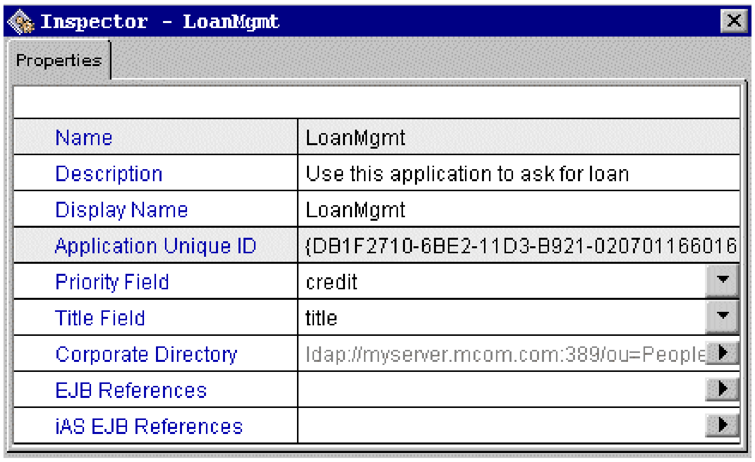
Process Manager is itself a J2EE module. If an EJB needs to interact with the Process Manager runtime or applications, its deployment descriptor must contain an `<ejb-ref>` tag. Similarly, if a Process Manager application uses an external EJB (via custom activity, custom field or in the JavaScript associated with any activities), it should contain the necessary `<ejb-ref>` tags in the deployment descriptor when the application is registered with iAS. If you use EJB references in an application, you must map these references to iAS EJB references.

Adding EJB References

1. Launch Process Manager Builder.
2. Open the application to which you need to add EJB references.
3. Right click on the application name in the application tree view and select Properties.

   The properties Inspector appears.

Figure 12-3 The properties Inspector.



Inspector - LoanMgmt

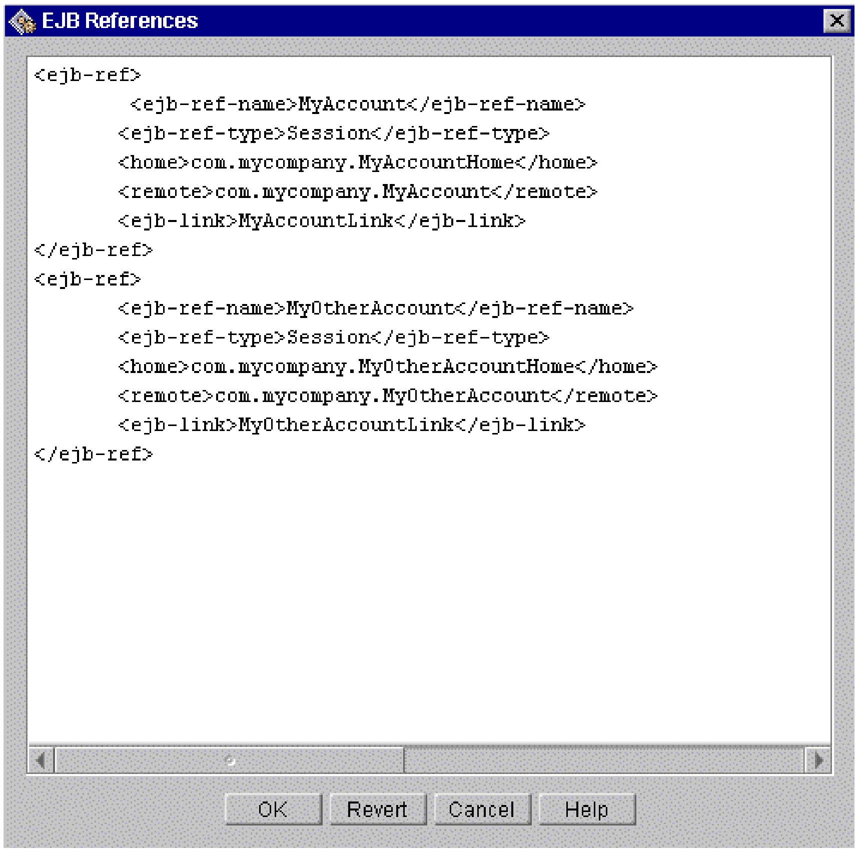
Properties

Name	LoanMgmt
Description	Use this application to ask for loan
Display Name	LoanMgmt
Application Unique ID	{DB1F2710-6BE2-11D3-B921-020701166016
Priority Field	credit
Title Field	title
Corporate Directory	ldap://myserver.mcom.com:389/ou=People
EJB References	
iAS EJB References	

4. Select the arrow corresponding to EJB References.

   The EJB References editor window appears.

Figure 12-4 The EJB References editor.



```
<ejb-ref>
         <ejb-ref-name>MyAccount</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.mycompany.MyAccountHome</home>
        <remote>com.mycompany.MyAccount</remote>
        <ejb-link>MyAccountLink</ejb-link>
</ejb-ref>
<ejb-ref>
        <ejb-ref-name>MyOtherAccount</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.mycompany.MyOtherAccountHome</home>
        <remote>com.mycompany.MyOtherAccount</remote>
        <ejb-link>MyOtherAccountLink</ejb-link>
</ejb-ref>
```

5. Enter the EJB references in the editor window.

The syntax is identical to J2EE deployment descriptor files. For example, if your PM application uses EJBs called MyAccount and MyOtherAccount, the EJB references might look like this:

Code Example 12-1 Example of an EJB reference

```
<ejb-ref>
        <ejb-ref-name>MyAccount</ejb-ref-name>
       <ejb-ref-type>Session</ejb-ref-type>
       <home>com.mycompany.MyAccountHome</home>
       <remote>com.mycompany.MyAccount</remote>
       <ejb-link>MyAccountLink</ejb-link>
</ejb-ref>
<ejb-ref>
       <ejb-ref-name>MyOtherAccount</ejb-ref-name>
       <ejb-ref-type>Session</ejb-ref-type>
       <home>com.mycompany.MyOtherAccountHome</home>
       <remote>com.mycompany.MyOtherAccount</remote>
       <ejb-link>MyOtherAccountLink</ejb-link>
</ejb-ref>
```

6. Select OK to save your edits.

   Once you have created and saved the EJB reference in the editor, the reference appears as an item in the drop-down list on the properties Inspector window.

   You can add as many EJB references as you need in the free form editor. Note that the text you enter is not validated. Anything you type is sent verbatim to the server and placed in the deployment descriptor of the PM application.

Adding iAS EJB References to a Process Manager Application

In addition to declaring EJB references in the deployment descriptor file, you must declare an `<ejb-ref>` tag that maps the `<ejb-ref-name>` declared in the `<ejb-ref>` tag to the name in its registry (via `<jndi-name>` tag). This is known as an iAS EJB reference. If you do not declare an iAS EJB reference, the iAS environment will not register the EJB reference.

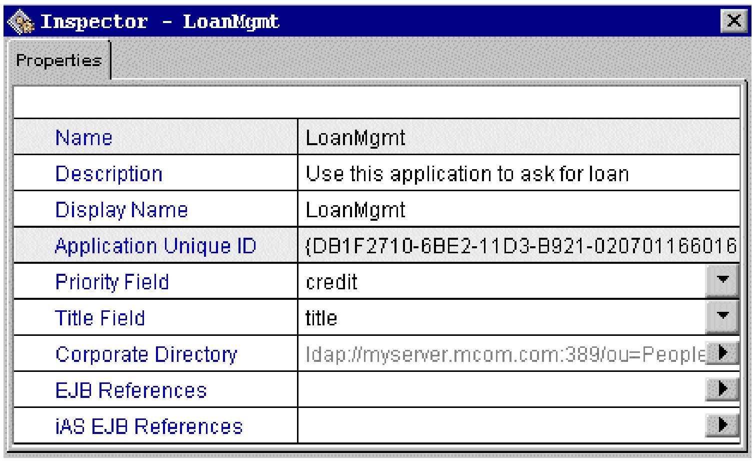
For more specific information about iAS EJB references, refer to the *iPlanet Application Server Java Programmers Guide*.

Adding iAS EJB References

1. Launch Process Manager Builder.
2. Open the application to which you need to add iAS EJB references.
3. Right click on the application name in the application tree view and select Properties.

   The properties Inspector appears.

Figure 12-5 The properties Inspector.



Inspector - LoanMgmt

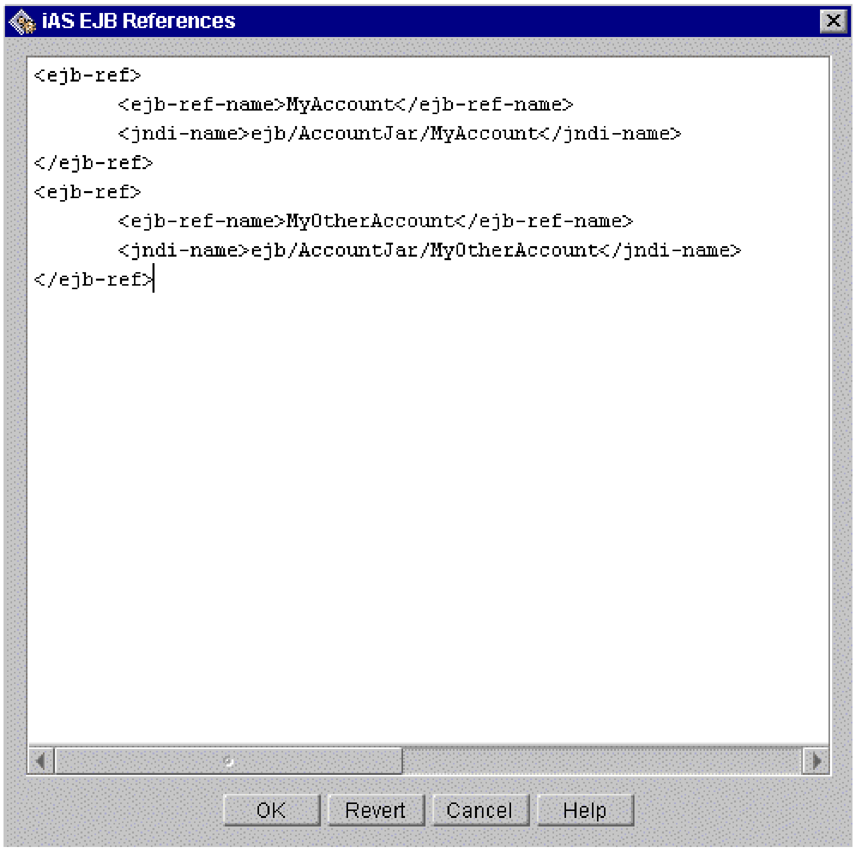
Properties

Name	LoanMgmt
Description	Use this application to ask for loan
Display Name	LoanMgmt
Application Unique ID	{DB1F2710-6BE2-11D3-B921-020701166016
Priority Field	credit
Title Field	title
Corporate Directory	ldap://myserver.mcom.com:389/ou=People
EJB References	
iAS EJB References	

4. Select the arrow corresponding to iAS EJB References.

   The iAS EJB References editor window appears.

Figure 12-6 The iAS EJB References editor.



```
<ejb-ref>
        <ejb-ref-name>MyAccount</ejb-ref-name>
        <jndi-name>ejb/AccountJar/MyAccount</jndi-name>
</ejb-ref>
<ejb-ref>
        <ejb-ref-name>MyOtherAccount</ejb-ref-name>
        <jndi-name>ejb/AccountJar/MyOtherAccount</jndi-name>
</ejb-ref>
```

5. Enter the iAS EJB references in the editor window.

The syntax is identical to J2EE deployment descriptor files. For example, if your PM application uses EJBs called MyAccount and MyOtherAccount, the iAS EJB mappings might look like this:

Code Example 12-2 Example of an iAS EJB Reference

```
<ejb-ref>
       <ejb-ref-name>MyAccount</ejb-ref-name>
       <jndi-name>ejb/AccountJar/MyAccount</jndi-name>
</ejb-ref>
<ejb-ref>
       <ejb-ref-name>MyOtherAccount</ejb-ref-name>
       <jndi-name>ejb/AccountJar/MyOtherAccount</jndi-name>
</ejb-ref>
```

6. Select OK to save your edits.

Once you have created and saved the iAS EJB reference in the editor, the reference appears as an item in the drop-down list on the properties Inspector window.

You can add as many EJB references as you need in the free form editor. Note that the text you enter is not validated. Anything you type is sent verbatim to the server and placed in the deployment descriptor of the PM application.

Adding EJB References to an Application Using the Command Line Deploy Tool

You can add EJB references to an application using the Command Line Deploy Tool. The Command Line Deploy Tool delivers the deployment functionality of the Process Manager Builder Deploy function without having to use the Builder's UI.

To learn how to use the Command Line Deploy Tool, see "Deploying Applications with the Command Line Deploy Tool" on page 216.

EJB references must be specified in the deployment descriptor (.dd) file when using the Command Line Deploy Tool. If you use this tool to register EJB references, you must specify all the EJB references on one single line following `ejbRefField` or `iasEjbRefField`. If you want to specify this on multiple lines, you need to escape each end of line with a \. The following example shows a typical EJB reference in the deployment descriptor file:

Code Example 12-3 Example of an EJB reference in a deployment descriptor (.dd) file

```
ejbRefField: <ejb-ref>\n \
  <ejb-ref-name>MyAccount</ejb-ref-name>\n \
  <ejb-ref-type>Session</ejb-ref-type>\n \
  <home>com.mycompany.MyAccountHome</home>\n \
  <remote>com.mycompany.MyAccount</remote>\n \
  <ejb-link>MyAccountLink</ejb-link>\n \
</ejb-ref>\n \
<ejb-ref>\n \
  <ejb-ref-name>MyOtherAccount</ejb-ref-name>\n \
  <ejb-ref-type>Session</ejb-ref-type>\n \
  <home>com.mycompany.MyOtherAccountHome</home>\n \
  <remote>com.mycompany.MyOtherAccount</remote>\n \
  <ejb-link>MyOtherAccountLink</ejb-link>\n \
</ejb-ref>\n \

iasEjbRefField: <ejb-ref>\n \
  <ejb-ref-name>MyAccount</ejb-ref-name>\n \
  <jndi-name>ejb/AccountJar/MyAccount</jndi-name>\n \
</ejb-ref>\n \
<ejb-ref>\n \
  <ejb-ref-name>MyOtherAccount</ejb-ref-name>\n \
  <jndi-name>ejb/AccountJar/MyOtherAccount</jndi-name>\n \
</ejb-ref>\n \
```

NOTE The \n in the above example instructs the system to insert a newline. This allows you to inspect the references in the Process Builder EJB References text editor at a later time. The text you enter is not validated. Anything you type is sent verbatim to the server and placed in the deployment descriptor of the PM application.

Chapter 13

Setting Up Searching

If you want end users to be able to search their process instance data, you need to design the application to enable the search capability.

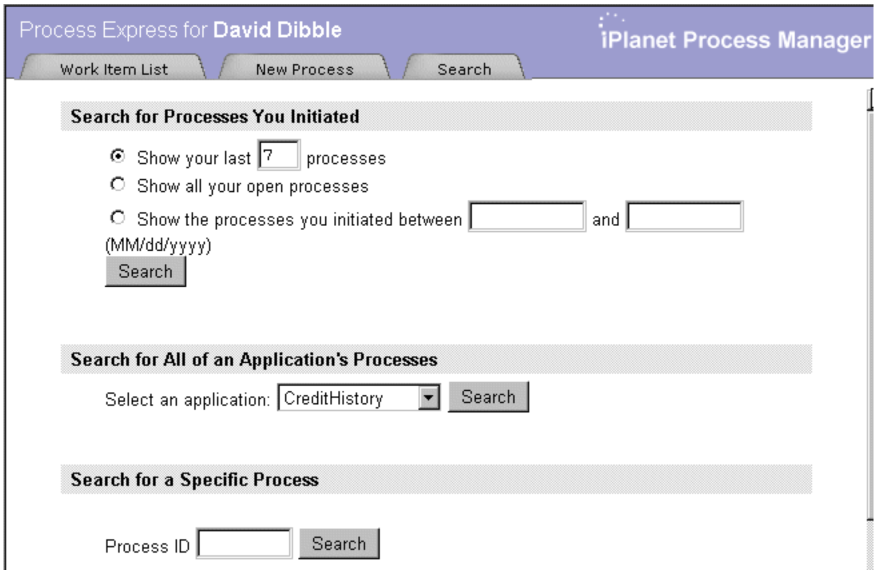
This chapter includes these sections:

- "Types of Searching" on page 259
- "Enabling Searching" on page 261

Types of Searching

You can make applications and data searchable if you want to allow end users to search applications in Process Express. In each application, you can configure two levels of searching: global searching and application-specific searching. The search page is similar to the one shown in Figure 13-1.

Figure 13-1 The Process Express Search Page


Process Express for **David Dibble**

iPlanet Process Manager

Search for Processes You Initiated

(●) Show your last [7] processes

( ) Show all your open processes

( ) Show the processes you initiated between [ ] and [ ] (MM/dd/yyyy)

Search for All of an Application's Processes

Select an application: CreditHistory

Search for a Specific Process

Process ID [ ]

In addition, the user can search by their request number, but you do not have to set up anything in the application to allow that kind of search. It is set up automatically for every application.

Global Searching

Global searching is always available to users when they search using the following options:

- the most recent process instances they created
- all the process instances they created that are still open
- all the process instances they created within a date range

Each search option generates a list of process instances that match its search criteria. Each entry in a list contains a link that displays the process instance and another link that displays the history of the process instance.

Application-Specific Searching

You can also configure an application so that participants can search for particular data within it. For example, if you have a data sheet application, you can configure it so that the user can search all instances of the data sheet process by product model number.

The Process Manager software automatically generates a search form for an application. The search form lets users search for values of fields if you made the fields searchable when you created the field. For example, if your application includes nine data fields, but you only made two searchable, the user can search for the values in those two fields.

The search form generates a list of process instances that match the search criteria. Each entry in the list contains a link to display the process instance and a link to the history of the process instance.

Enabling Searching

To set up searching for an application, you must do the following:

- Set up the appropriate groups to allow members to perform searches.
- Set up forms so that all users for whom you enable searching have forms at all activities.
- Set the appropriate fields to be searchable.

Allowing a Group to Search

To allow members of a group to perform searches, you must set the group to allow searching. If you do not allow searching, members of the group will not be able to use the search functionality to check the status of process instances or to find process instances related to specific criteria. You can allow searching for Application, Corporate, and Dynamic groups. If users are members of more than one group, they can search as long as one of the groups they belong to allows searching.

➤ **To allow members of a group to search**

1. In the application tree view, open the Groups and Roles folder.
2. Double-click a group icon, or highlight the group and click Inspect.
3. In the dialog box for the appropriate group, check the Allow Search checkbox.
4. Close the dialog box. Your changes are saved automatically.

Setting Up Forms

To allow users to get information about a specific step in a process instance, they must have a form they can access for that step. For example, if the creator of a process instance searches for information about it, and the process instance is at a step where there isn't a form the creator can access, he or she cannot get more information about the step beyond its listing in the search results. If the user has a form for that step, the listing has a link to view more information.

One way to make sure that there is always a way for users to see more information on a process instance is to create forms which give a status of the process and set them up for the "all" group. However, for sensitive work items, you might not want to allow all users to see information during the process. In those cases you would set up forms for users who need to have access to the data.

➤ **To allow users to see more than the search listing in the search results**

1. In the application tree view, double-click the Form Access icon, or highlight the icon and click Inspect.

   The Form Access window appears.

2. Scan the vertical columns to make sure every user you want to view the data has a form they can see for every activity.

   Note that you do not need to assign a form specifically for every role. For example, if you have created a form for the "all" role, the creator and other groups and roles will be able to access that form.

3. If you want to add a form to a column, drag the form from the Application Tree View window to the appropriate box in the Form Access window.

   If the form you need hasn't been created, you need to create one and come back to this step. You may have to create a new form if you have not created an appropriate form already.

4. Close the Form Access window.

Allowing Searching for Fields

In order for users to be able to use a field as a criterion when they search, you need to make the field searchable. The following predefined field types can be made searchable:

- Computed
- Date
- DateTime
- Radio Button
- Select List
- TextField
- UserPicker Widget

➤ **To make a field searchable**

1. In the application tree view, double-click the field you want to make searchable, or highlight it and click Inspect.
2. In the Inspector window, set the Allow Search property to true.
3. Close the window. Your changes are saved automatically.

NOTE If a field is searchable, the field length must not exceed 2000 characters.

If you have created your own field class IDs instead of using the predefined ones, you must include the Allow Search property in the JSB file. For details, see the Custom Data Fields chapter of the *Process Manager Programmer's Guide*.

Chapter 14

The Data Sheet Application

This chapter describes the Data Sheet sample application that is shipped with this product. It includes a general walkthrough of the Data Sheet application, and specific information on the sample application functions.

This chapter includes these sections:

Data Sheet Application Overview

This application automates the process of creating a product data sheet, including the following steps:

- uploading text files and graphics
- formatting the data sheet automatically
- approving the data sheet through two levels of management
- publishing on a web site

The application contains all the necessary forms, user roles, and database fields to complete the process.

NOTE If you want users to be able to digitally sign the approval form, you must use Netscape Communicator to run this application.

Data Sheet Process Map

The data sheet's process map, shown in Figure 14-1, graphically describes the application's steps.

The process map shows the following steps:

Entry Point The process map contains one entry point for creating a new datasheet.

Activities There are three activities: Add Art, Approve Pricing (Product Mgr) where the Product Manager approves pricing, and Approve Pricing (V.P), where the V.P. approves pricing.

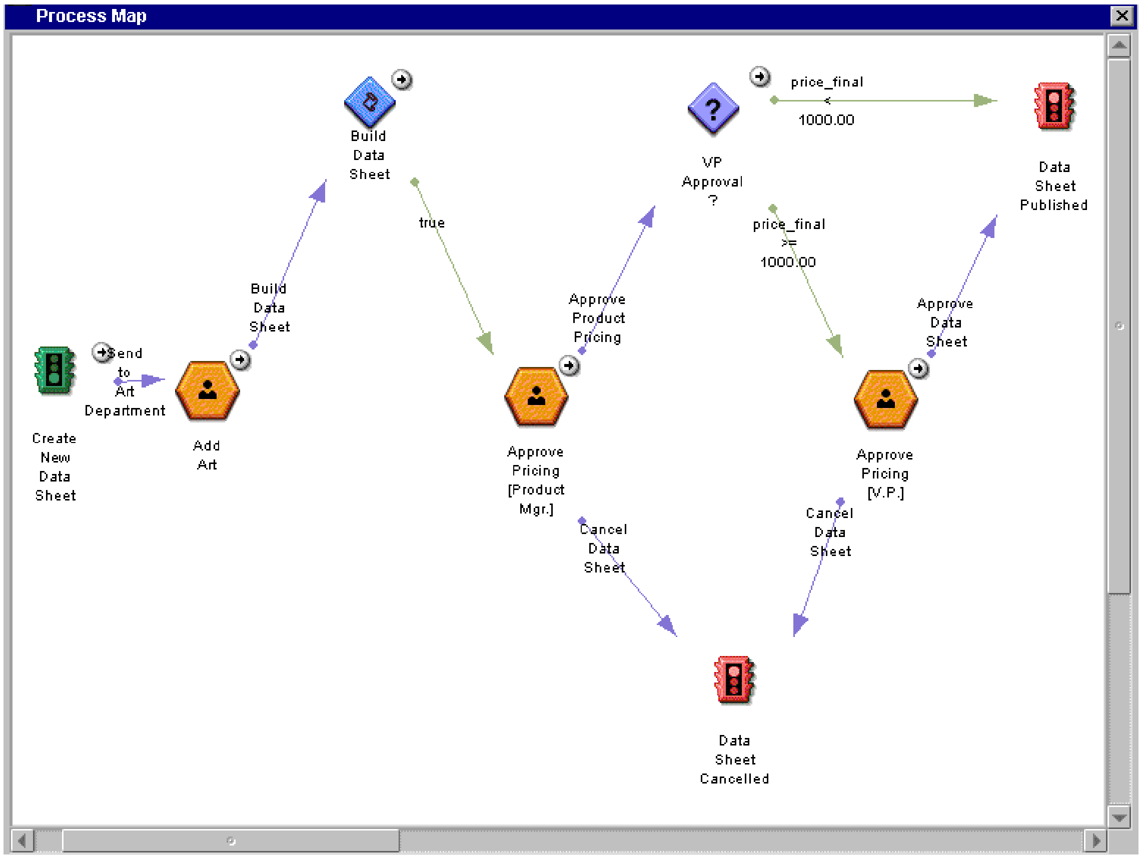
Automated Activity There is one automated activity, called Build Data Sheet, where the data sheet is built.

Decision Point There is one decision point, called VP Approval, which depends upon price. If the price is less than $1000 dollars, the release is approved. If the price is $1000 or more, the vice president needs to approve the data sheet before it can be released.

Exit Points There are two exit points in the Data Sheet application, Data Sheet Cancelled that occurs if the V.P. or Product Manager does not approve the data sheet, and Data Sheet Published that occurs if the data sheet passes all approvals and it is published.

Notification There is one notification in the data sheet application. It occurs partway through the process at the product manager approval step.

Figure 14-1 The data sheet process map



Data Sheet Walkthrough

This walkthrough takes you through the process described by the product datasheet application, describing the portions of the application involved in each step.

The Data Sheet Entry Point

Users initiate a data sheet at the Create New Datasheet entry point. When they initiated the data sheet in Process Express, they see the newDataSheet.html form. On this form users enter product and pricing information. The product information includes the model number, which is the title field. Process Express displays the title field to identify the process instance.

Note that if you are testing this application yourself, and you sign into Process Express as `admin`, then the creator who initiates the process is set to `admin`.

The following fields are on the newDataSheet.html form:

- Product Type (type)
- Model Number (model)
- Product Description (description)
- Base Price (price)
- VAR Discount (discount_code)

The product description needs to be in the form of an attached text file. The file is stored in the content store and used to create the data sheet.

When users have filled out the form, they click the Send to Art Department button to continue. The button takes its name from the name of the connecting line drawn on the process map between two steps.

The Title Field

The application has a title field, which shows up in Process Express wherever a particular process instance is identified. The process instance title is stored in a special data field, called title. It is also set in the application's properties.

➤ **To see the Title Field in the application's properties**

1. In the application tree view, right click the application name (DataSheet) and choose Properties, or double-click the name, or highlight it and click Inspect.

2. The application's properties appear. The Title Field property set to the "title" data field.

The value of the title field for a process instance is set by the entry point's completion script, `computeTitle`, which puts process instance information into the `title` field. For more information on the script, see "The computeTitle Script" on page 286.

Add Art Activity

The next step is to get the art from the art department. To see the properties for this activity, double-click the activity in the application tree view, or highlight the activity and click Inspect.

Assignment Script

For demonstration purposes, the data sheet activity is assigned to a user with the `toCreator` assignment script. This is one of the standard assignment scripts that ships with Process Builder. It assigns the work item to the creator of the process instance.

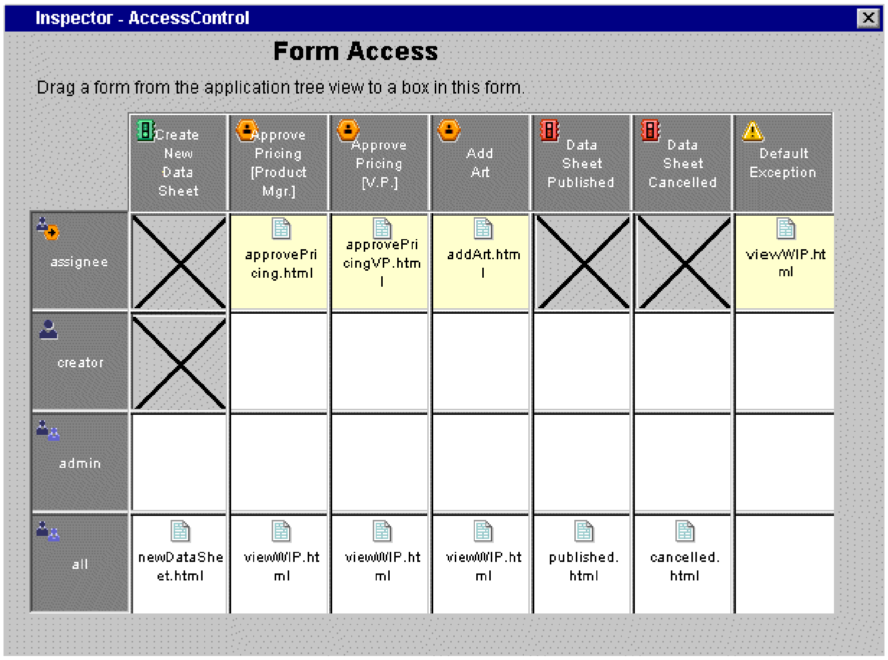
Expiration Setter Script

The work item for this step has an expiration date. It uses a hard-coded length of time (1 day) for the expiration date, set by a standard Process Builder script, `expireIn`. It takes the parameters of a number, and a unit of time (days, weeks, etc.). You do not select this script from a drop-down list (as the assignment script is selected). Instead, when you access the wizard for setting expiration dates, you have radio buttons for "Never expire," "Expire in," and "Defined by a script." If you choose "Expire in," there's a field for a number, and a drop-down list for the unit of time, which you use to set the parameters for the `expireIn` script.

The Forms

Two forms are visible to users at this activity. The assigned user sees the form addArt.html. All other users see the viewWIP.html (View Work in Progress) form. The various forms are assigned to different groups and users in the Form Access window. See Figure 14-2. You access this window from the application tree view, by double-clicking the Form Access icon.

Figure 14-2 The Data Sheet Form Access window



Inspector - AccessControl

Form Access

Drag a form from the application tree view to a box in this form.

	Create New Data Sheet	Approve Pricing [Product Mgr.]	Approve Pricing [V.P.]	Add Art	Data Sheet Published	Data Sheet Cancelled	Default Exception
assignee	✗	approvePricing.html	approvePricingVP.html	addArt.html	✗	✗	viewWIP.html
creator	✗						
admin							
all	newDataSheet.html	viewWIP.html	viewWIP.html	viewWIP.html	published.html	cancelled.html	

By reading the grid you can see what role or group sees what form at each step of the application. The groups and roles are in the order specified in the evaluation order in the Groups and Roles properties. For this example, you can see that at the Add Art activity, the assignee sees one form (addArt.html), while the “all” group sees another (ViewWIP.html).

On the addArt.html form, some fields are display-only, and some the assignee can edit. To see whether or not a field can be edited, select the field on the form and hold down the right mouse button. From there, select the Properties item, and you see a list of the properties of that field in that form. (the Data Dictionary item on this menu allows you to access field properties that are application-wide). The Display Mode property shows whether the field is set to EDIT, VIEW or HIDDEN.

When a field is set to EDIT, only the assignee can actually edit it. If another user views the form, they see the field in VIEW mode.

The only fields that the assignee can edit on this form are layout and art.

The other fields are view-only. The art field is a file attachment field, which means that to complete this field the user attaches a .gif file. The file is stored in the URL specified in the Content Store Properties window. After completing the form, users click the Build Data Sheet button.

On the viewWIP.html form, all fields are view-only. This form is available to "all" so everybody but the assignee sees it. One of the fields, stageURL, gives the URL for the current version of the data sheet, so anyone can monitor its progress.

The Automated Activity

After the user pushes the Build Data Sheet button, the next step is an automated activity called Build Data Sheet. Because this is an automated activity, no user is assigned, and it never becomes a work item on a user's work list.

To see the properties of the automated activity, in the application tree view, double-click the name of the activity, or highlight it and click Inspect.

When the process reaches the automated activity in the data sheet application, the action takes place immediately (the Schedule and Deferred properties are blank). In addition, the Completion Script property (where another script would be run after the automation script ended) is also blank. In your own applications, those fields can be used to schedule the time the activity takes place, and to perform some additional action when the automated activity is completed.

The Automation Script for this activity is called `buildDS()`. No automation scripts are included in the default script dictionary that is built when you create a new application. The `buildDS` script is unique to the data sheet application.

➤ **To look at the script**

1. In the application tree view, open the Script Dictionary.
2. Open the Automated Scripts folder.
3. Double-click `buildDS`, or highlight it and click Inspect. In the Inspector window, you can read and edit the actual text of the script. In addition, you can check the script's syntax to make sure it executes properly.

If the script returns `true` (that is, if it was successfully completed) the process moves on to the next step.

For more information on how the script works, see "The buildDS Script" on page 281.

The Approve Pricing (Product Manager) Activity

In this activity, the product manager approves the pricing on the data sheet.

Assignment Script

For demonstration purposes, the data sheet activity is assigned to a user with the `toCreator` assignment script. This is one of the standard assignment scripts that ships with Process Builder. It assigns the work item to the creator of the process instance. If you are testing this application as the `admin` user, the creator is set to `admin`.

In a non-demonstration situation, the parameter would either be set to the Product Manager's user ID, or a different assignment script could determine the vice president for the product using the corporate user directory and/or the data in the database.

Forms

There are two forms visible to users at this activity. The assignee sees the form approvePricing.html. All other users see the viewWIP.html form.

On the approvePricing.html form, the product manager can see all the pricing data and approve it. The product manager provides authentication by clicking the Sign It button. The field is signed_by_pm.

After signing the form, the manager clicks one of two buttons, Approve Product Pricing or Cancel Data Sheet. If the manager chooses Cancel Data Sheet, the data sheet is not published, and it reaches an exit point.

If the manager approves the price, the application moves on to the next step, the decision point.

In this sample, the application does not check to see if the product manager has signed the form. You could build this functionality into your own application by adding a client script to the form called `onSubmitForm()`.

Notification

At the Approve by Process step, a mail notification is sent to the person who started the process instance (the person who filled out the original form at the entry point to create a data sheet). If you are testing this application as the `admin` user, make sure that this user has an email address. For information about providing an email address, see "Adding an Email Attribute for a User" on page 293.

➤ **To see the notification's properties**

1. In the application tree view, open the Process Definition folder.
2. In the Process Definition folder, expand the Approve Pricing (Product Mgr.) activity.
3. You see three items, the two transitions (Cancel Data Sheet and Approve Product Data Sheet) and NotifyApproving. Double-click NotifyApproving, or highlight it and click Inspect to see the notification's properties.

In the application tree view, in the Pricing Approval by Process activity, there is a notification called NotifyApproving. If you double-click the notification, or highlight it and click Inspect, you see the notification's properties.

The Email Address(es) property instructs the application who to send the notification to. In this case, there is a script called `emailOfCreator()` that mails the message to the person who started the process instance. The message is contained in the Email Body property, which in this example contains some sample text.

The VP Approval Decision Point

If the manager approves the data sheet and price, the process takes a different path depending upon the price. If the price is $1000 or more, the vice president must approve it and it goes to the Approve Pricing (VP) activity.

If the price is less than $1000, the data sheet is released and published, and reaches an exit point.

The decision point looks at the value in the price_final field to determine the price.

➤ **To see the conditions of the decision point**

1. In the application tree view, in the Process Definition folder, expand the VP Approval ? decision point.
2. You see two transitions, priceOver1000 and priceUnder1000. Right click priceOver1000 and choose Properties.
3. The transition's properties appear, including the Condition property. The condition property is set to price_final >= 1000.00, which means that if the value of the price_final field is greater than or equal to 1000.00, this transition is the one that is used.

There is no form, since the decision point takes place automatically.

The Approve Price (VP) Activity

In this activity, the vice president approves the pricing on the data sheet.

Assignment Script

For demonstration purposes, the data sheet activity is assigned to a user with the `toCreator` assignment script. This is one of the standard assignment scripts that ships with Process Builder. It assigns the work item to the creator of the process instance.

In a non-demonstration situation, the parameter would either be set to the vice president's user ID, or a different assignment script could determine the vice president for the product using the corporate user directory and the data in the database.

Forms

There are two forms visible to users at this activity. The assigned user sees the approvePricingVP.html form. All other users see the viewWIP.html form.

On the approvePricingVP.html form, the vice president can see the pricing data and approve it. The vice president provides authentication by clicking the Sign It button. The field is signed_by_vp.

NOTE Digital signing is not supported by Microsoft Internet Explorer.

After signing the form, the vice president clicks one of two buttons, either Approve Data Sheet, or Cancel Data Sheet. If the V.P. selects Cancel Data Sheet, the data sheet does not get published, and it comes to an exit point.

If the manager selects Approve, the data sheet is released and published at the exit point.

Exit Points

This application has two exit points: Data Sheet Cancelled, and Data Sheet Published. If the product manager or vice president does not approve the pricing on the data sheet, the data sheet is canceled and the process ends at Data Sheet Cancelled.

If the product manger and the V.P. (if necessary) approve the pricing, then the data sheet is released and published and the process ends at that exit point.

The exit points each have a form associated with them, cancelled.html and published.html. These forms allow users to see whether the data sheet was published or not. They also have links to the data sheet on the server, so users can see the final product.

Groups and Roles

The following groups and roles are defined for this application:

- all
- creator
- admin

These are the default groups and roles that are created for every application.

Data Dictionary

The data dictionary contains all the data fields that are used by the application and stored in the database. Table 14-1 lists the data dictionary for the DataSheet sample application. To see the properties for these fields, do the following. In the application tree view, inside the Data Dictionary folder, double-click the name of the field.

Fields are defined to have a class ID, which in turn determines the properties the field has. For more information on the properties for specific field class IDs, see Chapter 7, "Defining Data Fields."

Table 14-1 Data Sheet Application Fields

Field	**Field Class ID**
approved_by_pm	computed
approved_by_vp	computed
art	file attachment
discount	textfield
discount_code	select list
discount_desc	textfield
description	file attachment

Table 14-1 Data Sheet Application Fields *(Continued)*

Field	Field Class ID
layout	select list
model	textfield
price	textfield
price_final	textfield
signed_by_pm	signature
signed_by_vp	signature
stageURL	URL
title	textfield
type	radio buttons

When you add fields, you can either add them with predefined class IDs, or you can create your own class IDs. All of the class IDs for fields in the data sheet application are predefined class IDs.

Forms

The Data Sheet application contains the following forms:

- newDataSheet.html
- addArt.html
- approvePricing.html
- viewWIP.html
- approvePricingVP.html
- cancelled.html
- published.html

The walkthrough contained information about what forms users see at what step of the application. To view this information, in the application tree view, double-click Form Access.

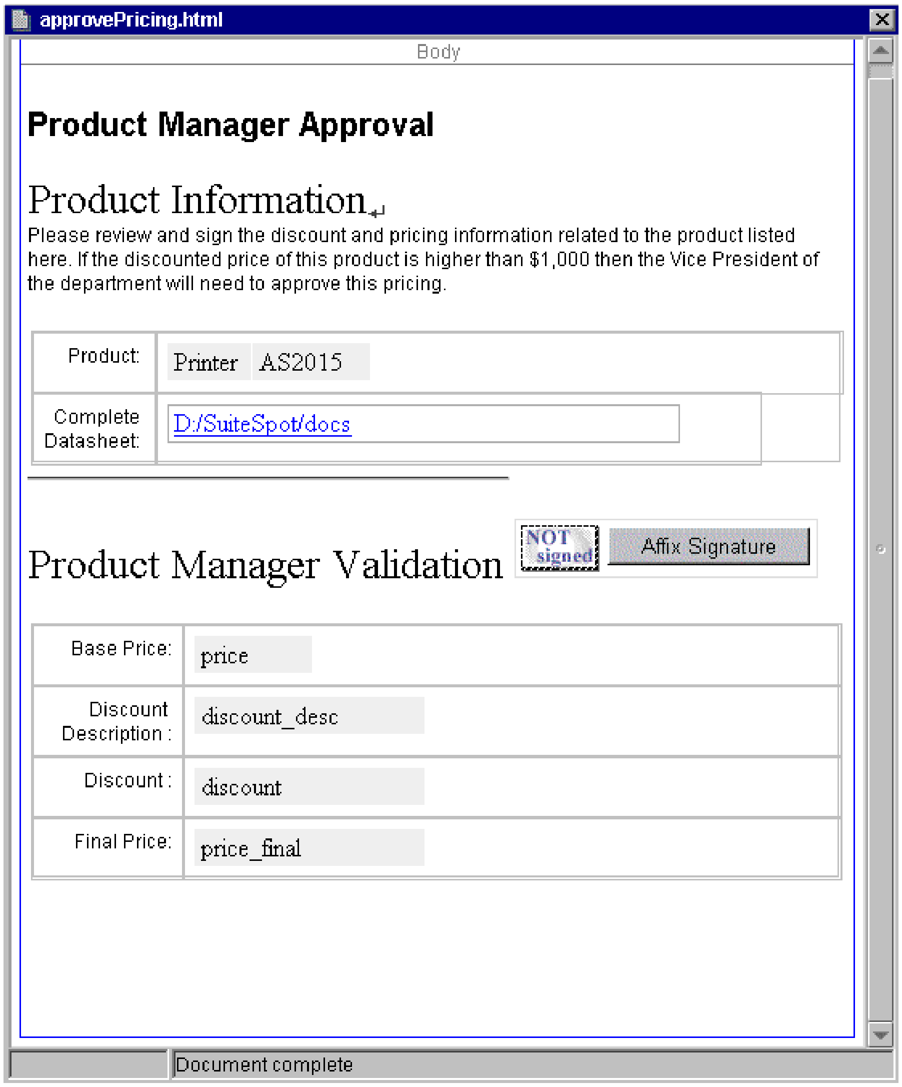
➤ **To view a form**

1. In the application tree view, double-click the Form Dictionary to open it.
2. In the Form Dictionary, open the form you want by double-clicking it. You see a portion of the form that you can edit.

In addition to the editable part of the form, the user will also see header information and buttons to press.

Figure 14-3 shows the approvePricing.html form as it appears in Process Builder.

Figure 14-3 The approvePricing.html form in Process Builder



Product Manager Approval

Product Information

Please review and sign the discount and pricing information related to the product listed here. If the discounted price of this product is higher than $1,000 then the Vice President of the department will need to approve this pricing.

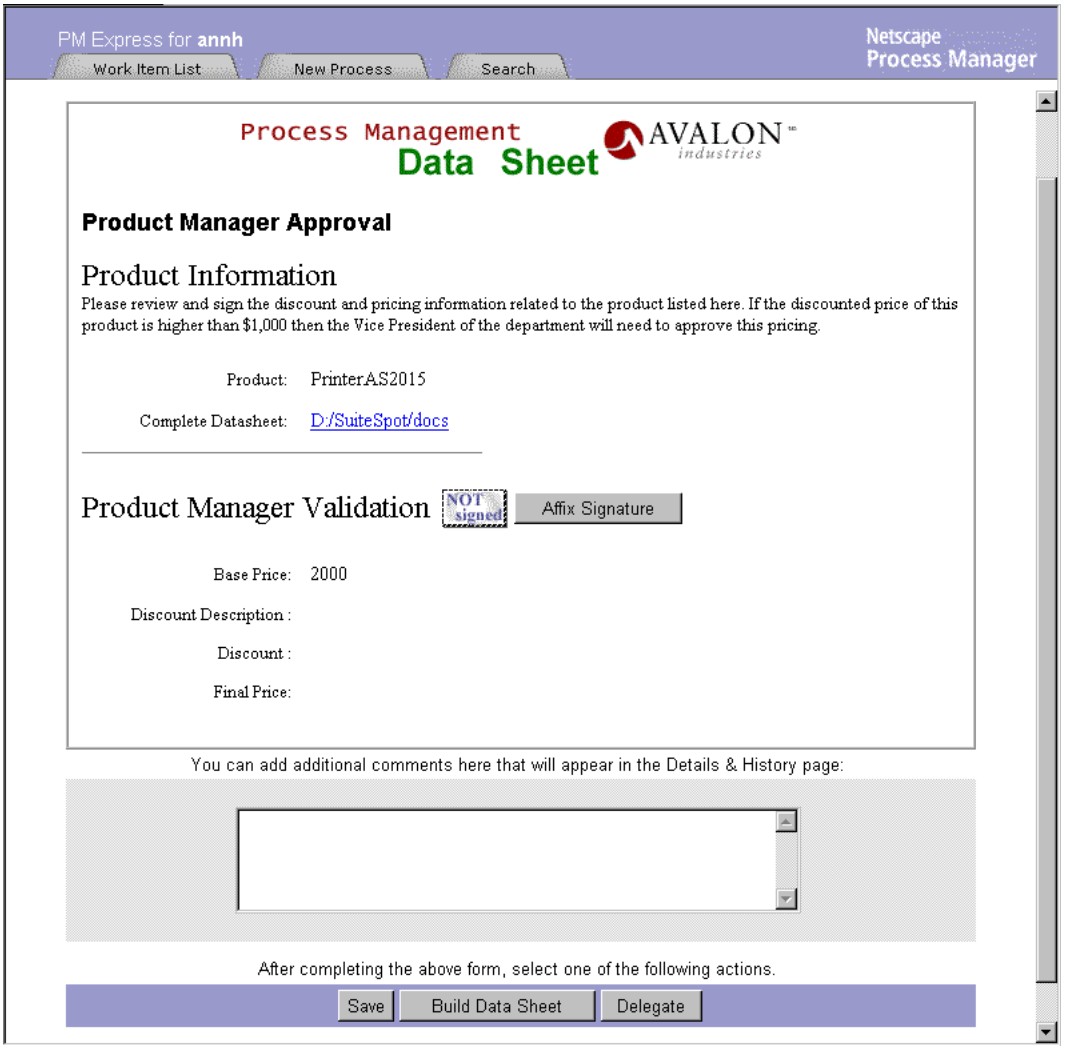
Product:	Printer AS2015
Complete Datasheet:	D:/SuiteSpot/docs

Product Manager Validation

Base Price:	price
Discount Description :	discount_desc
Discount :	discount
Final Price:	price_final

You can see only the body portion of the form. Figure 14-4 shows how the same form appears in Process Express.

Figure 14-4 The approvePricing.html form in Process Express



Product Manager Approval

Product Information

Please review and sign the discount and pricing information related to the product listed here. If the discounted price of this product is higher than $1,000 then the Vice President of the department will need to approve this pricing.

Product:	PrinterAS2015
Complete Datasheet:	D:/SuiteSpot/docs

Product Manager Validation

Base Price:	2000
Discount Description :	
Discount :	
Final Price:	

You can add additional comments here that will appear in the Details & History page:

After completing the above form, select one of the following actions.

The form in Process Express has a banner and buttons added to it. In addition, a form may also have a section for comments. The comments are controlled by the Activity’s Allow to Add Comments property. If the property is set to true, the comments section appears. Otherwise, it does not appear on the form.

The banner is a file called `banner.gif` found in the application's images folder (`builder/Applications/DataSheet/images`). The banner is created automatically when you create an application, and attached automatically when the form is displayed in Process Express. In your own applications, to change the banner, replace the `banner.gif` file with your own banner file. The file must still be called `banner.gif`.

The buttons are created using the transition names from the process map.

Script Dictionary

The script dictionary contains all the scripts that exist by default in any Process Builder application, and also the additional ones included in the data sheet application.

The data sheet application contains the following scripts in addition to the standard ones:

- `buildDS` (automation script)
- `lookupCode` (toolkit script)
- `buildDataSheet` (toolkit script)
- `computeTitle` (completion script)

These scripts demonstrate how to create scripts of some of the types you will need to create in your own application. They also show how to take advantage of the global scripts, for example, `getProcessInstance()`, which returns a `ProcessInstance` object which can be used to retrieve information about the process instance.

The buildDS Script

The automated activity that builds the database uses `buildDS`. If you double-click `buildDS` in the application tree view, you see the text of the script.

```
// Building the datasheet
// 1. LookUp discount information
// 2. Create and publish datasheet
function buildDS()
{
  return lookupCode()&&buildDataSheet();
}
```

This script calls other scripts in the toolkit. You add scripts to the toolkit if you want to be able to reuse them within the application.

This script executes each of the following scripts:

- `lookUpCode ()` -- sets the final price.
- `buildDataSheet ()` -- builds the datasheet as an HTML file.

The use of the `&&` operator means that the functions are evaluated in order left to right, and so long as each function returns `true`, the next function is evaluated. If all the functions return `true`, the statement returns `true`. If any of the functions return `false`, the remaining functions will not be evaluated, and the function returns `false`.

The lookupCode Script

The `lookupCode` script computes the value of the `price_final` field using the price and discount code that the user entered.

```
function lookupCode (){
  // The global function getProcessInstance()
  // returns the process instance.
  var pi = getProcessInstance();
  // The methods getData and setData of the process instance
  // are used to read the data field and set the value of the
  // datafield. When setting a value, you have to provide the
  // right JavaScript type.

  // Retrieve the discount_code.
  var discount_code= pi.getData("discount_code");
  // discount_code is an element of the data dictionary.
  // Construct discount description using discount_code.
  var desc = "Preferred VAR discount (code " + discount_code + ")"
;

  // The discount_code is something like ’10 A’.
  // Extract the numerical discount from the discount_code.
  // Note that parseInt(’10 A’) returns 10.
  var disc = parseInt(discount_code);
  // If the discount is not a number (isNaN)
  // add an entry in the error log and return false.
  if (isNaN(disc)){
      var e = new Object();
      e.inFunction        = "lookUpCode";
      e.discountCode      = discount_code;
      e.computedDiscount  = disc;
      logErrorMsg("ERROR_DISCOUNT_IS_NOT_VALID",e);
      // Return false to ask engine to rollback the transaction.
      return false;
  }
  // Update the process instance data by adding
  // discount description, discount rate, and final price.
  pi.setData("discount_desc", desc);
  pi.setData("discount",disc.toString() );
  pi.setData("price_final", Math.round(
        parseFloat(pi.getData("price"))*(100-disc))/100);

  // Return true because the script was successfully performed.
  return true ;
}
```

The buildDataSheet Script

This script creates a data sheet as an HTML file. It gets the data from the type, model, price, and discount code values that the user entered in the newDataSheet.html form at the entry point to this process. Additionally, the script opens a file that contains a description of the item in question, and adds the description to the data sheet. The script uses an HTML template file as the template for the datasheet, and ultimately publishes the datasheet as an HTML file.

The HTML template contains place holders identified by `$$` to indicate places where values of data fields should be plugged into the template. For example, the place holder `$$model$$` will be replaced by the value of the `model` data field.

For a sample of the `description.txt` file, see "The description.txt File" on page 288. For a sample of the `printer.html` template file, see "The printer.html Template File" on page 289.

```
function buildDataSheet (){
  // Use the global function getProcessInstance to get a
  // reference to the process instance.
  var pi = getProcessInstance();

  // Get the value of the description data field, which is a URL
  // pointing to where the description text file is located.
  var descriptionURL = pi.getData("description");

  // Use the global function getContentStore to get a reference
  // to the content store.
  var cStore = getContentStore();

  // Read the content of the description file.
  var descriptionContent = cStore.getContent(descriptionURL);

  // Get the path name of the application.
  var appPath = getApplicationPath();

  // Create a connection to the file containing the template
  // to be used to format the datasheet. The filename is
  // something like appPath/models/printer.html
  var template = new File (appPath + "models/" +
      pi.getData("type") + ".html");

  // Get the model.
  var model    = pi.getData("model");
```

```
// Construct the URL where the datasheet will be published.
// The global function getBaseForFileName returns the base
// URL of the location in the content store reserved for the
// process instance.
var dsURL = getBaseForFileName(pi.getInstanceId()) + model
    + ".html";

// Create the string to hold the content for the datasheet.
var dsContent = "<!-- Content of the dataSheet -->";

// Open the template in read-only mode. If the file is not
// opened successfully, log an error and return false.
if (!template.open("r")){
    template.close();
    var e       = new Object();
    e.model     = model;
    e.template  = template;
    e.stagingURL= dsURL;
    logErrorMsg("ERROR_COULD_NOT_OPEN_TEMPLATE_FOR_READ",e);
    return false ;
}

// Merge the template with data field values with the content
// in the description.txt file. Values of data fields are
// plugged into corresponding placeholders in a template.
// For example, $$model$$ in the template is replaced by the
// value of the model data field.
// Paragraphs in the description.txt file are substituted
// into the template in place of $$n$$ placeholders.
// For example, $$2$$ in the template is replaced by the
// second paragraph in description.txt.
var partArray  = descriptionContent.split("----");
  while (!template.eof()){

  // Read a line from the template file and split it into
// strings using $$ as the separator.
// Create an array of the strings. param[0] is the first
// string, param[1] is the second string, etc.
  // For example:
  // <TD WIDTH="30%"><FONT SIZE=+1>$$$price_final$$</FONT></TD>
  // param[0] = <TD WIDTH="30%"> <FONT SIZE=+1>$
  // param[1]= price_final
  // param[2] = </FONT></TD>
  var line  = template.readln();
  var param = line.split("$$");
```

```
    // If there are 3 sections, substitute either a description
    // paragraph or a datafield value for the middle string.
    if (param.length == 3)
  {
     var ind = parseInt(param[1]);
     // Returns 0 if not an integer.
     // If ind is > 0 get a paragraph from description.txt
     // else get a data field value.
     var subthis = ind > 0 ? partArray[ind-1]
pi.getData(param[1]);

     // Write the first string into the datasheet.
    dsContent += param[0];
     // Write the substituted string into the datasheet.
     dsContent += subthis;
     // Write the last string into the datasheet.
     dsContent += param[2];
    }
    // If there are not 3 strings, it means the line contains no
    // placeholders, so write the whole line into the datasheet.
    else {
      dsContent += line;
    }
    dsContent += "\n";
  }



  // Close the template.
  template.close();

  // Use the store method of the contentStore object to
  // publish the datasheet to the destination URL
  // that was defined earlier.
  var status = cStore.store(dsContent,dsURL);

  // Keep track of where the datasheet has been published by
  // storing the URL in a data field.
  pi.setData("stageURL",dsURL);
  return true;
}
```

The computeTitle Script

This script is invoked every time a data sheet is initiated. It is responsible for initializing the title of the process instance. The title of the process instance shows up in the work list when the application is running in production mode.

This script relies on the fact that the application contains a field called `title` that is used as the title field for the application (that is, in the application's properties, the title property is set to title.)

Users enter the values for the type and model during the entry point. This script runs as a completion script for the entry point. The script simply gets the value of the type and model data fields, and combines them to specify the name of the process instance. For example, if the type is printer and the model is 2345, the title of the process instance is "printer 2345".

```
function computeTitle(){
  // Use the global function getProcessInstance()
  // to get a reference to the process instance.
  var pi = getProcessInstance();

  // Read the value of the field "model" which was set by the
  // participant in the newDataSheet form in the entry point.
  var model = pi.getData("model");

  // Read the value of the field "type" which was set by the
  // participant in the newDataSheet form in the entry point.
  var type  = pi.getData("type");

  // Set the value of the field "title" to be an aggregation
  // of the type and the model.
  pi.setData ("title", type + " " + model);

  // The execution has been successful.
  return true;
}
```

Content Store

The Content Store has information about where attached files are stored on the web server. Before deploying the application, you must fill in the URL, Public User, and Public Password properties. For more information on configuring the Data Sheet sample application for deployment, see "Configuring the Data Sheet Application" on page 292. For more information on content store properties, see "Setting Up the Content Store" on page 168.

For details about configuring the content store for use with a secure web server, see "Enabling Content Store Over https" on page 48.

Finished Data Sheet Example

The data sheet uses two files that the user adds during the process. These files reside in the `builder/Samples/DataSheet/demo` folder:

- `description.txt`
- `image.gif`

There are also some files shipped with the sample application in the `builder/Samples/DataSheet/models` folder:

- `bar.gif`
- `Printer.html`
- `thumb_printer.gif`

When these files are combined by the application they create a data sheet. The following examples show a sample `description.txt` file, a sample `image.gif` file, and the finished data sheet using these files and the files in the `/models` folder.

The description.txt File

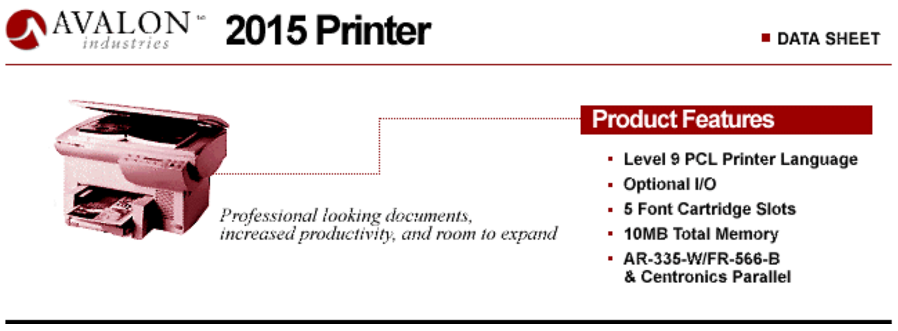
The user uploads a text file as part of completing the form to initiate a new data sheet. Here is an example:

```
Start with superior, high-resolution laser print quality,
versatile paper handling and 12 page-per-minute print speed
driven by a fast processor. What you get is the Aurora LaserJet
2015 printer, <B>Super Fast</B>delivering outstanding value for
individual or shared use from the market leader in laser printing
for the office.
----

The 12 page-per-minute engine and fast processor get performance
off to a great start. With the instant-on fuser that eliminates
warm-up time, you get a quick 19-second first page out speed.
Printing efficiency is further increased with a multi-sheet
feeder.
----

Megabytes of memory effectively doubled with Aurora’s Memory
Enhancement technology (MEt), and two parallel ports for seamless
network compatibility and functionality.
----

Professional, high-quality printing. Outstanding performance:
Compatible, Reliable Aurora printers with the best Image Quality
and High Speed printing on the Paper types you want.&nbsp;

<P>True 600 x 600 dpi resolution delivers clear lines, blacker
blacks and perfect gradients in all your text and graphics.
Brilliant graphics with at least 128 levels of gray for smoother
gray transitions and photorealistic images
```

The image.gif File

The user also adds artwork to the data sheet in the Add Art activity. Figure 14-5 shows an example of the kind of file a user might add:

Figure 14-5 Sample image.gif


AVALON industries
2015 Printer
DATA SHEET
Product Features
Level 9 PCL Printer Language
Optional I/O
5 Font Cartridge Slots
10MB Total Memory
AR-335-W/FR-566-B & Centronics Parallel
Professional looking documents, increased productivity, and room to expand

This image appears at the top of the finished data sheet.

The printer.html Template File

The printer template file is in the `builder/Samples/DataSheet/models` folder. The data sheet application uses it to format the data sheet. The HTML template file contains placeholders identified by $$ to indicate places where datafield values or paragraphs from the `description.txt` will appear.

For example, the placeholder `$$model$$` will be replaced by the value of the `model` data field. The place holders indicated as $$1$$, $$2$$, and so on, indicate places where paragraphs in the `description.txt` file will be plugged into the template. For example, $$1$$ indicates where the first paragraph of the description will be placed in the data sheet.

The HTML template must have no more than one placeholder per line. Any lines that do not contain placeholders are used in the datasheet without modification.

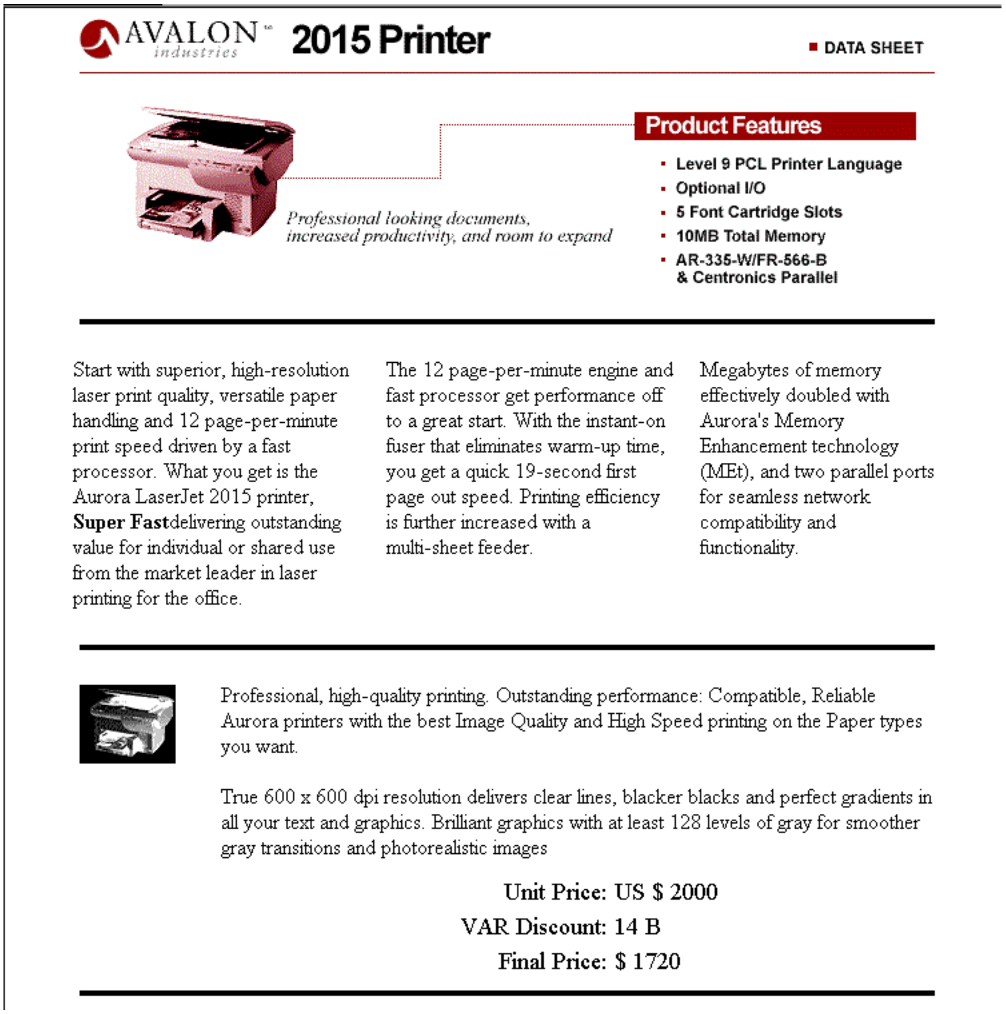
```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">

  <TITLE>$$model$$
      $$type$$ Data Sheet</TITLE>
</HEAD>
<BODY BGCOLOR="white">
<CENTER>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=10 WIDTH="650">
<TR ALIGN=LEFT VALIGN=TOP>
<TD COLSPAN=3 VALIGN=CENTER HALGIN=CENTER><IMG SRC="$$art$$"></TD>
</TR>
```

```
<TD ALIGN=LEFT VALIGN=TOP>
$$1$$
</TD>
<TD ALIGN=LEFT VALIGN=TOP>
$$2$$
</TD>
<TD ALIGN=LEFT VALIGN=TOP>
$$3$$
</TD> </TR>
</TABLE>

<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=10 WIDTH="650">
<TR><TD COLSPAN=2><IMG SRC="/wfDataSheet_b2/models/bar.gif"></TD></TR>
<TD VALIGN=TOP><IMG SRC="/wfDataSheet_b2/models/thumb_printer.gif"
ALIGN=LEFT></TD>
<TD>
$$4$$
</TD> </TR>
</TABLE>
<TABLE BORDER=0 CELLPADDING=2 WIDTH="650">
<TR>
  <TD WIDTH="70%" ALIGN=RIGHT><FONT SIZE=+1>Unit Price:</FONT></TD>
  <TD WIDTH="30%"><FONT SIZE=+1> US $ $$price$$ </FONT> </TD>
</TR>
<TR>
  <TD WIDTH="70%" ALIGN=RIGHT><FONT SIZE=+1> VAR Discount:</FONT></TD>
  <TD WIDTH="30%" ><FONT SIZE=+1>$$discount_code$$</FONT></TD>
</TR>
<TR>
  <TD WIDTH="70%" ALIGN=RIGHT><FONT SIZE=+1>Final Price:</FONT></TD>
  <TD WIDTH="30%"> <FONT SIZE=+1>$ $$price_final$$</FONT></TD>
</TR>
<TR><TD COLSPAN=2><IMG SRC="/wfDataSheet_b2/models/bar.gif"></TD></TR>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

The Finished Data Sheet

When the data sheet is built, all the pieces come together. Figure 14-6 shows a sample data sheet which uses the `description.txt` file and the `image.gif` file in earlier examples.

Figure 14-6 The finished data sheet



AVALON industries **2015 Printer** ▪ DATA SHEET

Product Features

- Level 9 PCL Printer Language
- Optional I/O
- 5 Font Cartridge Slots
- 10MB Total Memory
- AR-335-W/FR-566-B & Centronics Parallel

Professional looking documents, increased productivity, and room to expand

Start with superior, high-resolution laser print quality, versatile paper handling and 12 page-per-minute print speed driven by a fast processor. What you get is the Aurora LaserJet 2015 printer, **Super Fast**delivering outstanding value for individual or shared use from the market leader in laser printing for the office.

The 12 page-per-minute engine and fast processor get performance off to a great start. With the instant-on fuser that eliminates warm-up time, you get a quick 19-second first page out speed. Printing efficiency is further increased with a multi-sheet feeder.

Megabytes of memory effectively doubled with Aurora's Memory Enhancement technology (MEt), and two parallel ports for seamless network compatibility and functionality.

Professional, high-quality printing. Outstanding performance: Compatible, Reliable Aurora printers with the best Image Quality and High Speed printing on the Paper types you want.

True 600 x 600 dpi resolution delivers clear lines, blacker blacks and perfect gradients in all your text and graphics. Brilliant graphics with at least 128 levels of gray for smoother gray transitions and photorealistic images

Unit Price: US $ 2000

VAR Discount: 14 B

Final Price: $ 1720

The pricing information comes from information entered by users into forms and approved by the product manager and the vice president, if required.

Configuring the Data Sheet Application

Before you can deploy the Data Sheet application on your own system, you must configure the application. The main steps are as follows:

1. Set the corporate directory as described in "Setting Your Corporate Directory" on page 91.
2. Make sure that your `admin` user has an email address. For details on creating an email address for the `admin` user, see "Adding an Email Attribute for a User" on page 293.
3. Configure the content store. For details, see "Using File Attachments and Content Stores" on page 294.

For demonstration purposes, all the activities are assigned to the creator of the process instance. As a result, if you deploy this application as currently set up, you can create a process instance and participate in all the steps.

Configuration Hints

This section describes the following configuration hints:

- "How Users Access the Data Sheet"
- "Process Express and iPlanet Application Server"

How Users Access the Data Sheet

In the DataSheet application, make sure that the `stageURL` data field points to the right URL. For example, if you point your content store at `http://kimba.iplanet.com/CS`, then the `stageURL` field must point to a folder on that server, such as `http://kimba/docs`. Note that `docs` is the default primary document directory for an iPlanet Web Server.

In addition, when you build a sample data sheet in Process Express, PAE stores the data sheet in a folder specific for each process instance. For example, suppose you have a default Windows NT installation. If your process is titled "AS2015-test" and its ID is 58, then the new data sheet is stored here:

```
C:\iplanet\servers\docs\CS\DataSheet\58\AS2015-test.html
```

The URL displayed to end users in Process Express is then:

```
http://kimba/CS/DataSheet/58/AS2015-test.html
```

Process Express and iPlanet Application Server

If you have iPlanet Application Server installed on the same system as Process Express, you will not be able to use Netscape Communicator with file attachments. To avoid this situation, you can install iAS and Process Express on different systems, or you can use Microsoft Internet Explorer to access Process Express.

For additional information about using the File Attachment data field, see "File Attachment" on page 155.

Adding an Email Attribute for a User

Some sample applications require a user to have an email account. For example, both the TimeOffRequest and DataSheet applications require the `admin` user to have email. If this user doesn't have email, you will get an error in the logs.

➤ **To add email for a user**

1. Launch iPlanet Console.

   On Windows NT, you can do this by using the Start/Programs menu command and looking for the iPlanet Server Products menu item.

2. In the authentication dialog box, enter this information:

   - administrative user name
   - administrative password
   - administration URL for the new directory server's Administration Server, including the port number

3. Click the User and Groups tab.
4. Enter "admin" or "adm*" in the user field, then click Search.
5. Select the admin user ID from the list of users displayed. By default, this is the iAS Administrator user.
6. Click Edit.
7. Enter an email account. If you are the administrator yourself, and you are just testing the system, enter your own email address to see what email notifications you can receive.
8. Click OK.
9. Close the iPlanet Console.

Using File Attachments and Content Stores

If you want to deploy and use the DataSheet application (or other applications that use file attachments), you must set a new access control list (ACL) for the content store. You must then define the content store properties in Process Builder. In summary, you must perform the following main steps:

- "Step 1: Administer the Web Server"
- "Step 2: Edit the Access Control List"
- "Step 3: Assign the Content Store's Style"
- "Step 4: Optionally, Add a Directory to the Web Server"
- "Step 5: Set the Values for the Application's Content Store"

These steps are described in detail in the sections that follow.

Step 1: Administer the Web Server

You set up an access control list (ACL) through the iPlanet Web Server. To administer the iPlanet Web Server, perform the following steps:

1. Go to the Server Administration page for your iPlanet Web Server.
2. Click the button for your iPlanet Web Server instance. You'll go to its Server Manager pages.
3. Click the Configuration Styles button.
4. Click the New Style link.
5. Enter `ContentStore` as the Style Name field and click OK.
6. In the page that appears, click the "Edit this style" button.
7. Click the Restrict Access link. This displays the Access Control List Management page.
8. Make sure that "the style ContentStore" is displayed, then click the "Edit Access Control" button. This displays the Access Control user interface.

Step 2: Edit the Access Control List

Within the Access Control user interface, perform the following steps:

1. Click the New Line button twice. This creates two new ACL lines.
    - Leave the first line unchanged.
    - In the second line, click Deny. A second pane appears in the lower part of the page.
2. Click Allow in the lower pane.
3. Click Update to update the top pane's values.
4. In the second line, click "anyone".
5. In the lower pane, click the "Authenticated people only" radio button.
6. Click the "All in the authentication database" radio button.
7. Make sure that the Default authentication method is selected.
8. Under "Authentication Database" at the bottom of the pane, select your corporate directory from the dropdown list, then click the radio button next to it.
9. Click Update in this frame to update the top pane.
10. In the top frame, click Submit to set the change.
11. Click OK to save your changes.
12. Click Save and Apply to apply your changes to the server.

Step 3: Assign the Content Store's Style

After editing the ACL, assign the content store's style by performing the following steps:

1. Under Configuration Styles, click the Assign Style link.
2. Enter `CS/*` as the URL prefix wildcard. This causes the URL to be: `http://`<*yourWebServer*>`/CS/*`
3. Select ContentStore from the drop down list and click OK.
4. Click OK again to save your changes.
5. Click Save and Apply to apply your changes to the web server.

Step 4: Optionally, Add a Directory to the Web Server

You can optionally add a directory to your web server to contain the files in the content store as described below. If you do not create the directory, Process Manager creates it for you (so long as you have privileges to create directories on the web server) when you deploy the application:

1. Using Windows Explorer, the command line, the operating system, the `mkdir` command or whatever other way you choose, create a new directory called `CS` in the web server's primary document directory. Typically, the primary document directory is /iplanet/servers/docs/, in which case you would create the directory /iplanet/servers/docs/`CS`.

Step 5: Set the Values for the Application's Content Store

After working with iPlanet Web Server (as described in Steps 1, 2, 3 and 4), you must make changes in Process Builder. You must set the content store values by performing the following steps:

1. If necessary, start Process Builder.
2. Locate the ContentStore entry in the application tree view.
3. Open the Inspector window for the ContentStore entry.
4. Enter a URL with the format `http://`<*yourWebServer*>`/CS`.
5. Enter the administrative user and password for the iPlanet Web Server. For example, enter `admin` and `admin`.

For more information on content store properties, see "The Content Store Inspector Window" on page 168.

Chapter 15

The Office Setup Application

This chapter describes the Office Setup sample application that is available with Process Manager.

This chapter contains the following sections:

Office Setup Application Overview

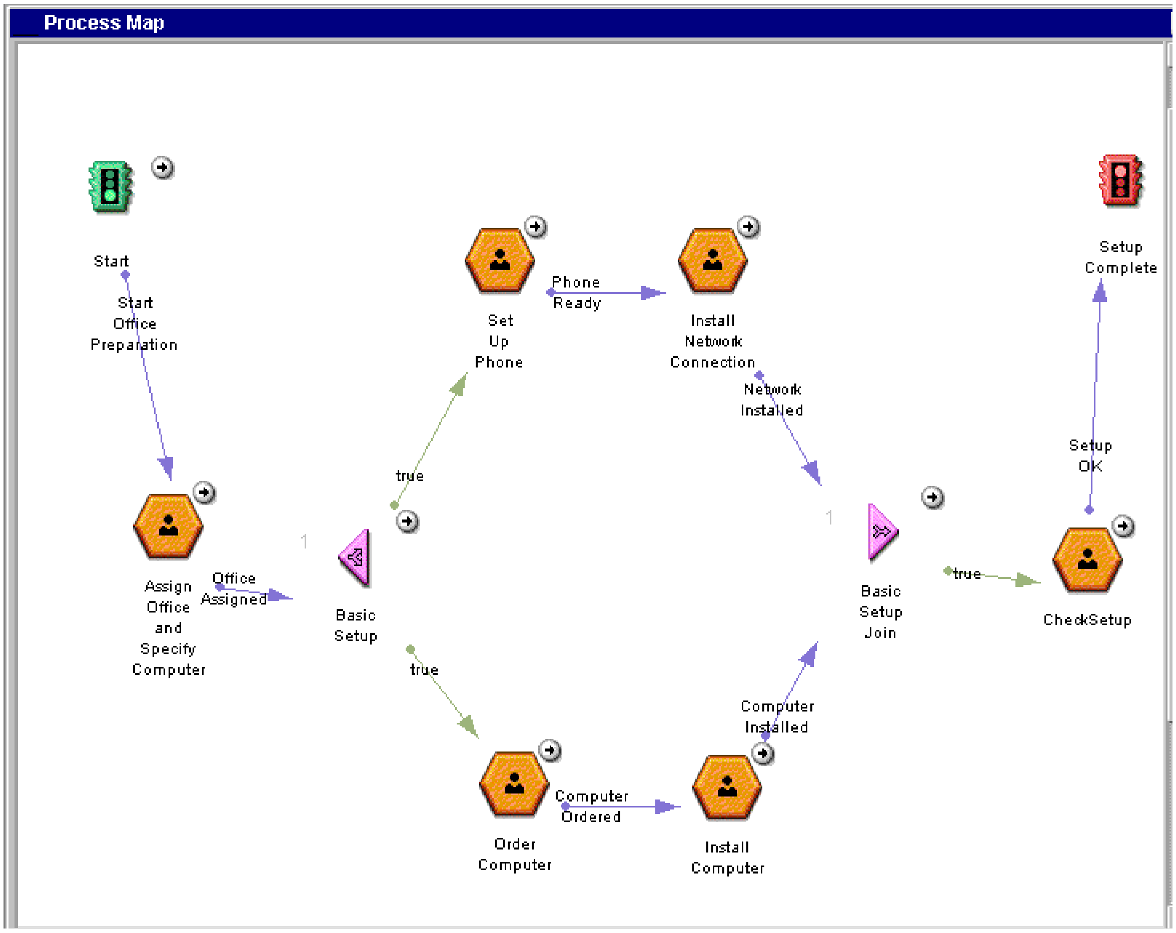
This application controls the process of setting up an office for a new employee. This application illustrates parallel processing.

This application contains a number of tasks that different people must perform to get an office ready. Some of the tasks must be carried out sequentially; for example, the purchasing department must order the computer before the MIS department can install it. Some of the tasks can be carried out in parallel; for example, the MIS department can set up the phone while the purchasing department is ordering the computer.

Office Setup Process Map

The process map for the Office Setup application is shown in Figure 15-1.

Figure 15-1 Process Map for the Office Setup application



Office Setup Walkthrough

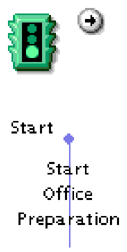
This section walks through the complete process, which contains the following entry points, activities (work items), splits, joins, and exit points:

- "Start Entry Point"
- "Assign Office and Specify Computer Work Item"
- "Basic Setup Split"
- "Set Up Phone Work Item"
- "Install Network Connection Work Item"
- "Order Computer Work Item"
- "Install Computer Work Item"
- "Basic Setup Join"
- "CheckSetup Work Item"
- "Setup Complete Exit Point"

Start Entry Point

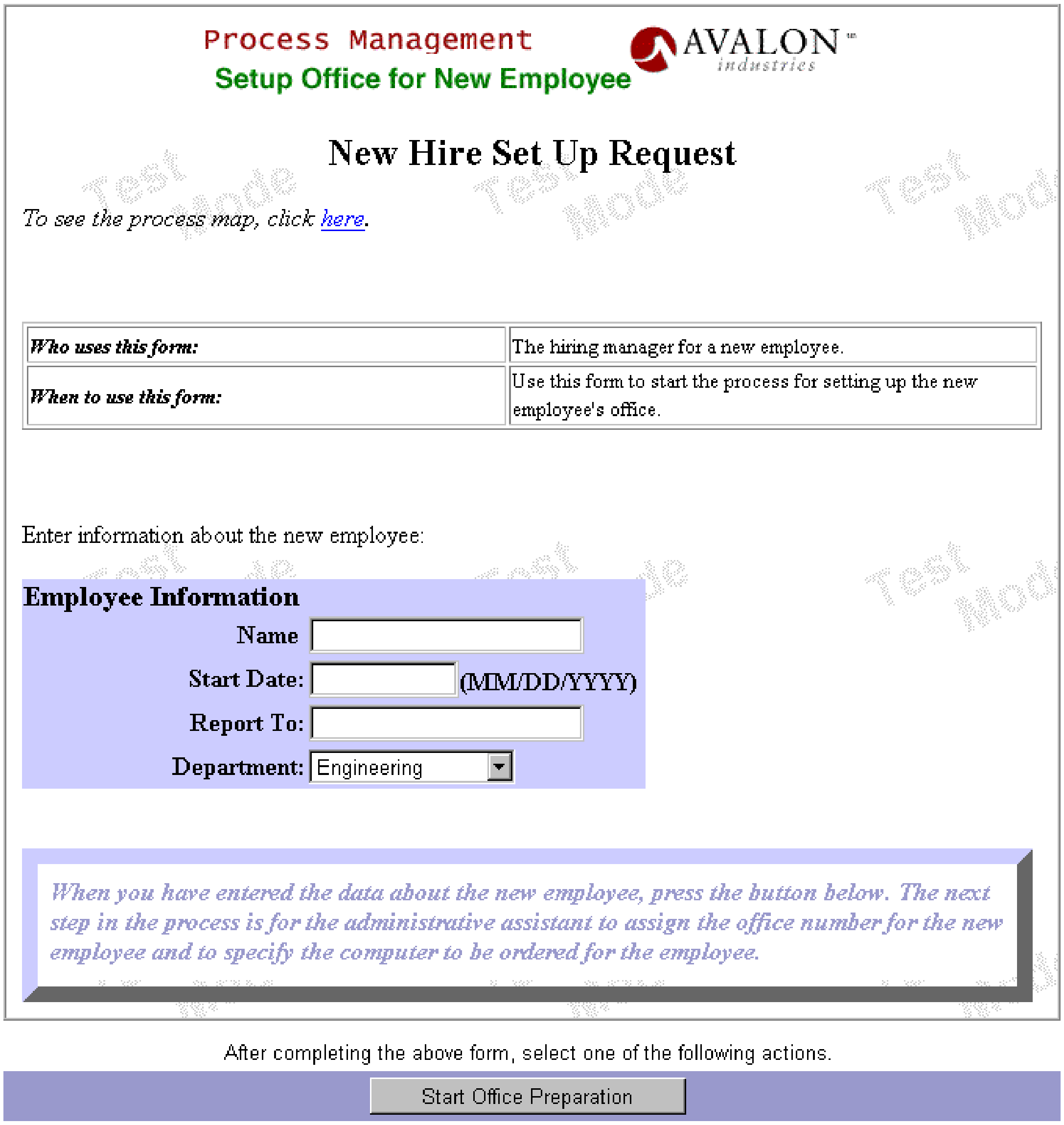
The process begins at the Start entry point, as shown in Figure 15-2:

Figure 15-2 The Start entry point



To start the process, the hiring manager fills in an entry point form specifying the new employee's name and start date in the form for the Start entry point, `prepareForNH.html`. This form is shown in Figure 15-3:

Figure 15-3 The prepareForNH.html form



New Hire Set Up Request

To see the process map, click here.

Who uses this form:	The hiring manager for a new employee.
When to use this form:	Use this form to start the process for setting up the new employee's office.

Enter information about the new employee:

Employee Information

Name	
Start Date:	(MM/DD/YYYY)
Report To:	
Department:	Engineering

When you have entered the data about the new employee, press the button below. The next step in the process is for the administrative assistant to assign the office number for the new employee and to specify the computer to be ordered for the employee.

After completing the above form, select one of the following actions.

This form sets the values of the following data fields:

dfEmpName The name of the new employee.

dfStartDate The new employee's start date. This must be in the format MM/DD/YYYY.

dfMgr The hiring manager.

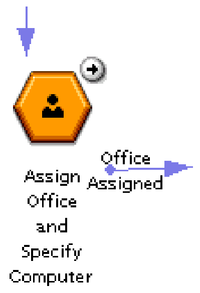
dfDeptName The department that the new employee will be joining. This datafield is a SELECT data field with a menu offering the choices Engineering, Human Resource, Marketing, and Sales.

This entry point has a completion script, `setRequesterField`, that sets the value of the `dfRequesterName` data field based on the process instance creator.

Assign Office and Specify Computer Work Item

The next step is for the manager's administrative assistant to assign an office and specify a computer, as shown in Figure 15-4:

Figure 15-4 The Assign Office and Specify Computer work item



The administrative assistant enters the office location and computer for the new employee, as shown in Figure 15-5. This form sets the values of the following data fields:

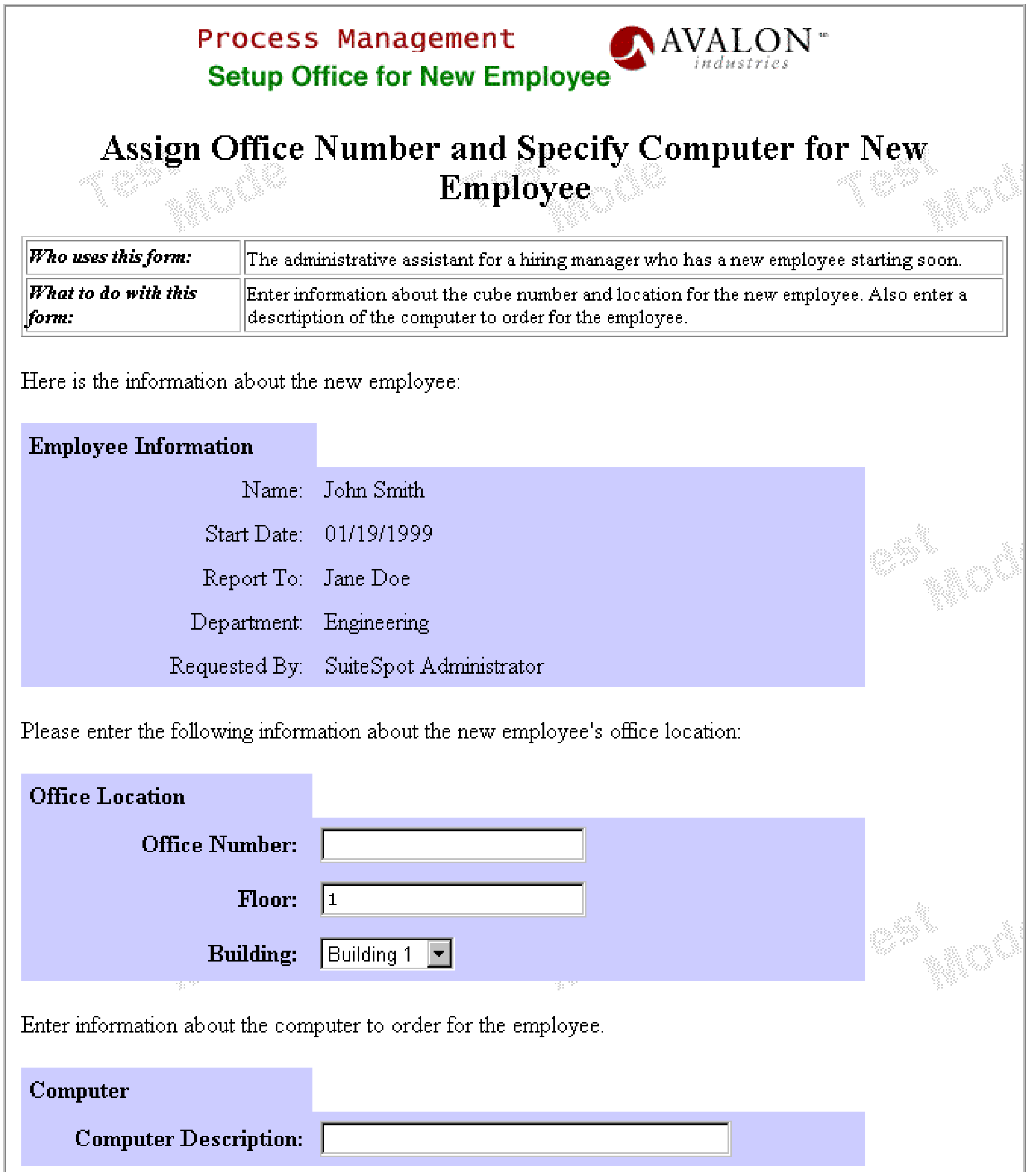
dfCubeNo The new employee's office number.

dfFloor The floor containing the office.

dfBldg The building where the office is located.

dfComputer A description of the computer to order for the new employee.

Figure 15-5 The assignCube.html form



Assign Office Number and Specify Computer for New Employee

Who uses this form:	The administrative assistant for a hiring manager who has a new employee starting soon.
What to do with this form:	Enter information about the cube number and location for the new employee. Also enter a descrtiption of the computer to order for the employee.

Here is the information about the new employee:

Employee Information

Name:	John Smith
Start Date:	01/19/1999
Report To:	Jane Doe
Department:	Engineering
Requested By:	SuiteSpot Administrator

Please enter the following information about the new employee's office location:

Office Location

Office Number:	
Floor:	1
Building:	Building 1

Enter information about the computer to order for the employee.

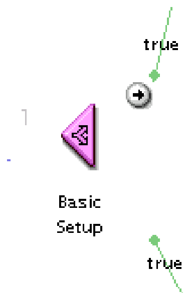
Computer

Computer Description:	

Basic Setup Split

At the Basic Setup split, the process splits into two parallel branches, as shown in Figure 15-6:

Figure 15-6 The Basic Setup split



One branch controls the process of installing the phone and network connection. The other branch controls the process of ordering and installing the computer.

The split has two conditional transitions leading off from it. As with decision points and automated activities, you can specify conditions to determine whether a route leading from a split is followed or not. By default, the condition for all routes leading out of a split are set to true, so that they are all active.

When you place a split icon on the process map, the icon splits and appears on the map in two halves. The split part appears where you placed it, and the corresponding join icon appears to the right. The distance between the split and join icons allows for the insertion of two activities between them. When you place the split icon, you may need to scroll your process map to the right to bring the join icon into view.

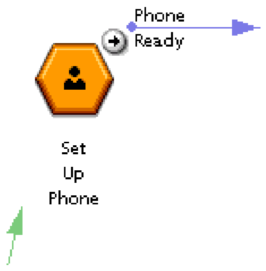
Each route leading away from a split must ultimately end at the corresponding join. It cannot go to an exit point or to a join for another split. Each branch of the split represents a series of activities that are be performed in parallel and ultimately merge back into a single path.

A split icon does not represent an activity, so there are no forms for it.

Set Up Phone Work Item

In the network installation branch, the first step is for the MIS department to assign a phone number and set up the phone. The process map for this step is shown in Figure 15-7:

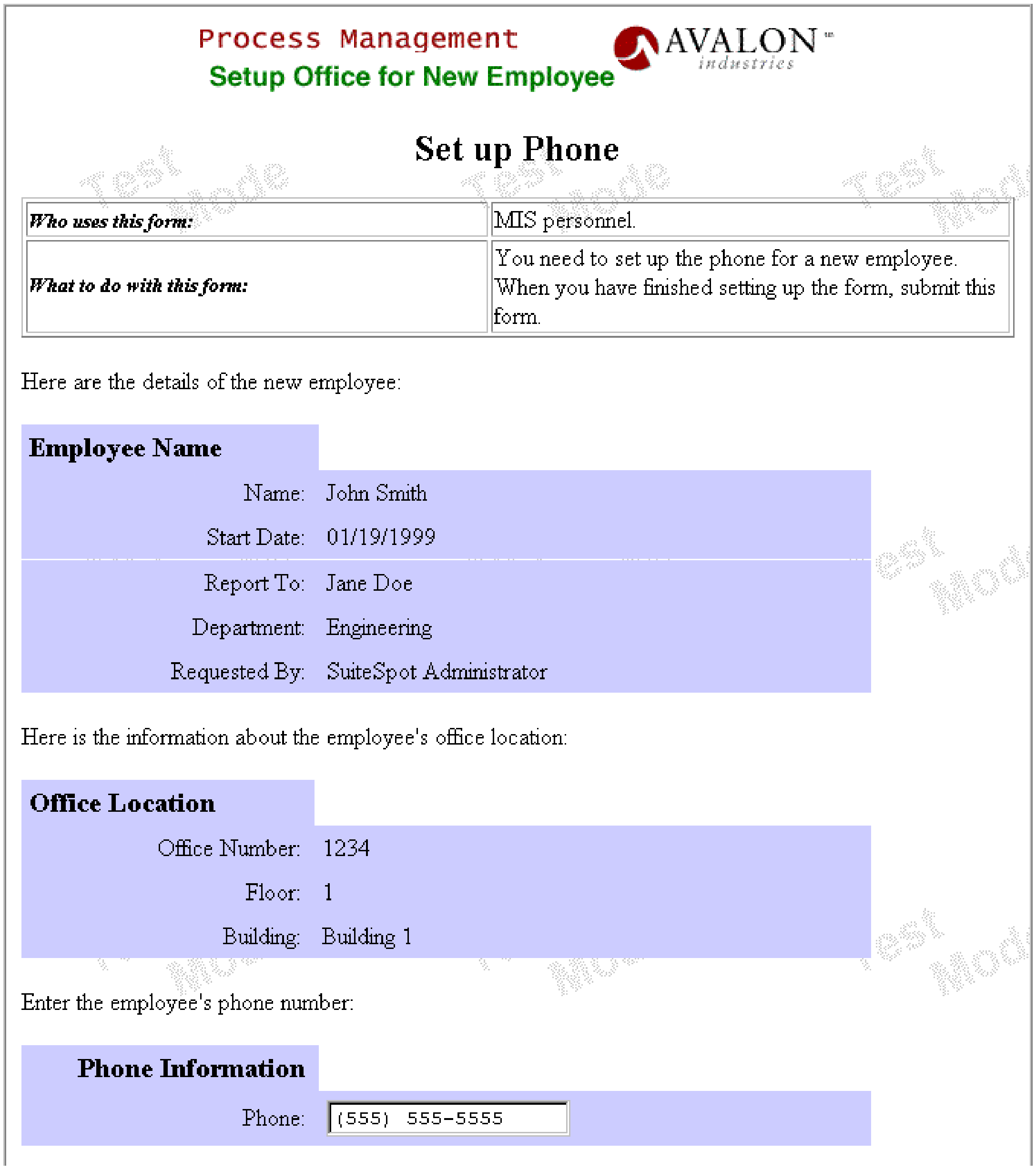
Figure 15-7 The Set Up Phone work item



The MIS person uses the `setupPhone.html` form, as shown in Figure 15-8. This form sets the values of the following data field:

dfPhone	The new employee's phone number.

Figure 15-8 The setupPhone.html form



Process Management
Setup Office for New Employee

AVALON industries

Set up Phone

Who uses this form:	MIS personnel.
What to do with this form:	You need to set up the phone for a new employee. When you have finished setting up the form, submit this form.

Here are the details of the new employee:

Employee Name

Name:	John Smith
Start Date:	01/19/1999
Report To:	Jane Doe
Department:	Engineering
Requested By:	SuiteSpot Administrator

Here is the information about the employee's office location:

Office Location

Office Number:	1234
Floor:	1
Building:	Building 1

Enter the employee's phone number:

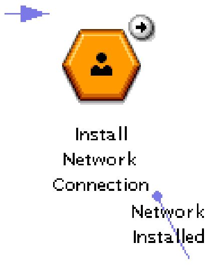
Phone Information

Phone:	(555) 555-5555

Install Network Connection Work Item

The next step, also done by the MIS department, is to install the network connection, making sure that the office has network cables and hardware. The process map for this step is shown in Figure 15-9:

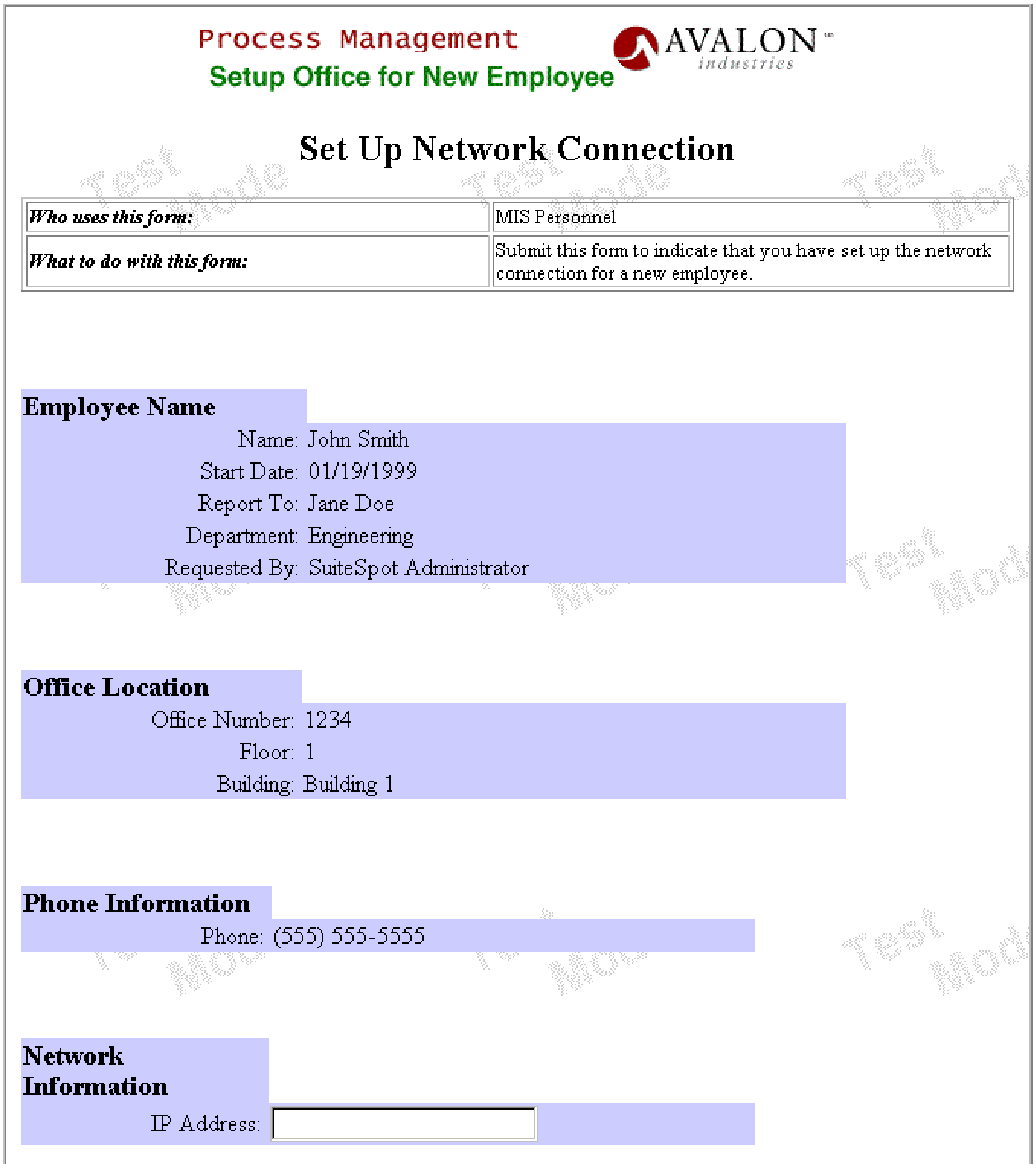
Figure 15-9 The Install Network Connection work item



The MIS department uses the `setupNetwork.html` form for this work item, as shown in Figure 15-10. This form sets the values of the following data field:

dfNetworkAddress	The network address.

Figure 15-10 The setupNetwork.html form



Set Up Network Connection

Who uses this form:	MIS Personnel
What to do with this form:	Submit this form to indicate that you have set up the network connection for a new employee.

Employee Name

Name:	John Smith
Start Date:	01/19/1999
Report To:	Jane Doe
Department:	Engineering
Requested By:	SuiteSpot Administrator

Office Location

Office Number:	1234
Floor:	1
Building:	Building 1

Phone Information

Phone:	(555) 555-5555

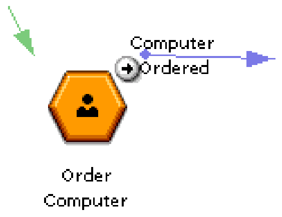
Network Information

IP Address:	

Order Computer Work Item

There is no need to wait for the phone and network setup to be finished before ordering and installing a computer in the office. Thus a parallel branch of the process controls the purchase and installation of the computer. In the computer installation branch, the first step is for someone in the purchasing department to order a new computer. The process map for this step is shown in Figure 15-11:

Figure 15-11 The Order Computer work item

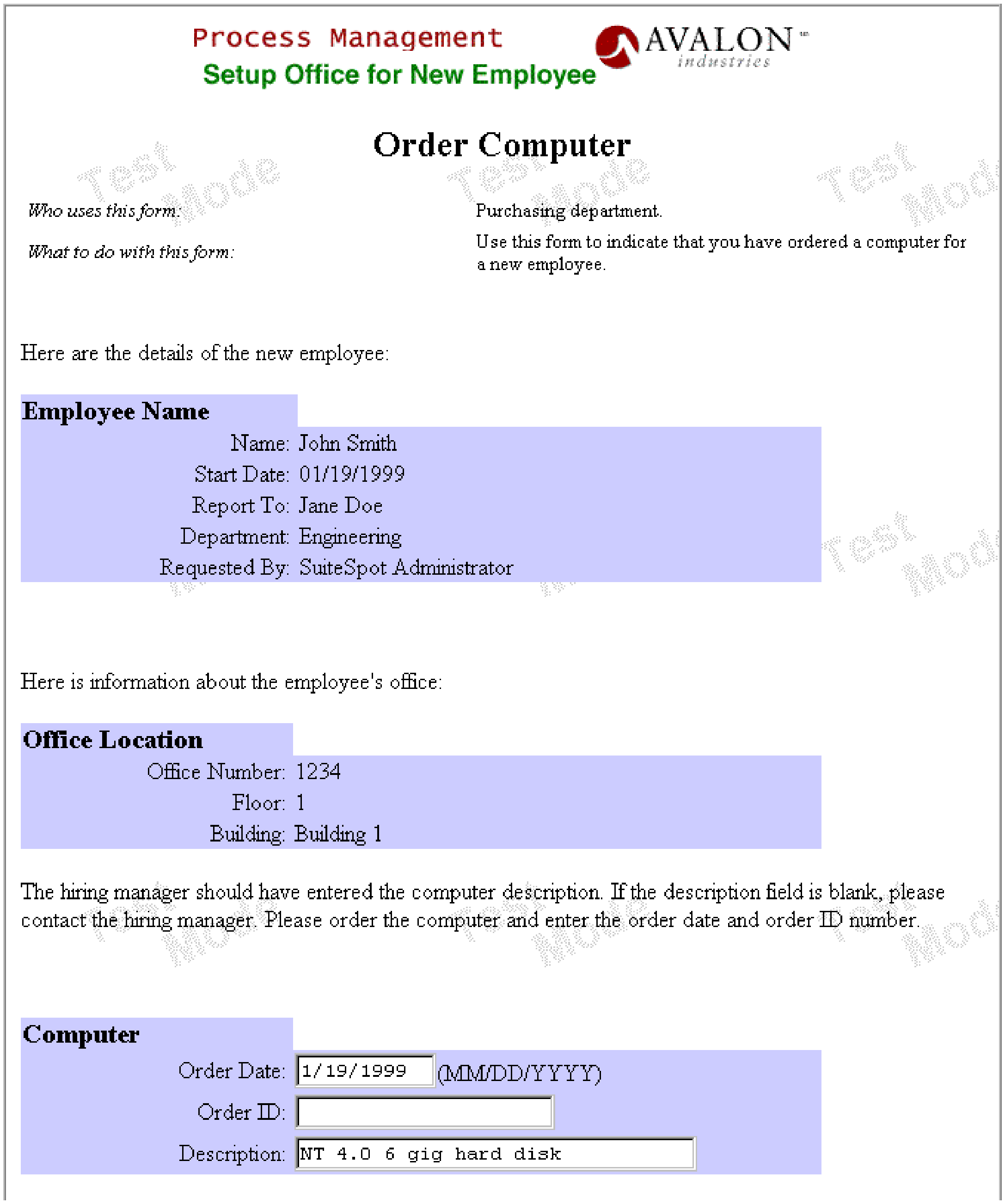


This work item has the form `orderComputer.html`, as shown in Figure 15-12. This form sets the values of the following data fields:

dfCompOrderDate The date on which the computer was ordered. This value must be a date such as 12/12/1998. This value is set by a client-side script, but the person ordering the computer can overwrite it. For more information on the client-side script, see "Embedded Client-Side Script" on page 322.

dfCompOrderID The ID or PO number for the order.

Figure 15-12 The orderComputer.html form



Order Computer

Who uses this form:	Purchasing department.
What to do with this form:	Use this form to indicate that you have ordered a computer for a new employee.

Here are the details of the new employee:

Employee Name

Name:	John Smith
Start Date:	01/19/1999
Report To:	Jane Doe
Department:	Engineering
Requested By:	SuiteSpot Administrator

Here is information about the employee's office:

Office Location

Office Number:	1234
Floor:	1
Building:	Building 1

The hiring manager should have entered the computer description. If the description field is blank, please contact the hiring manager. Please order the computer and enter the order date and order ID number.

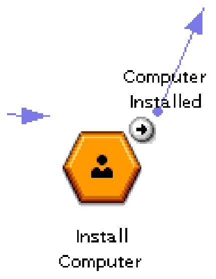
Computer

Order Date:	1/19/1999 (MM/DD/YYYY)
Order ID:	
Description:	NT 4.0 6 gig hard disk

Install Computer Work Item

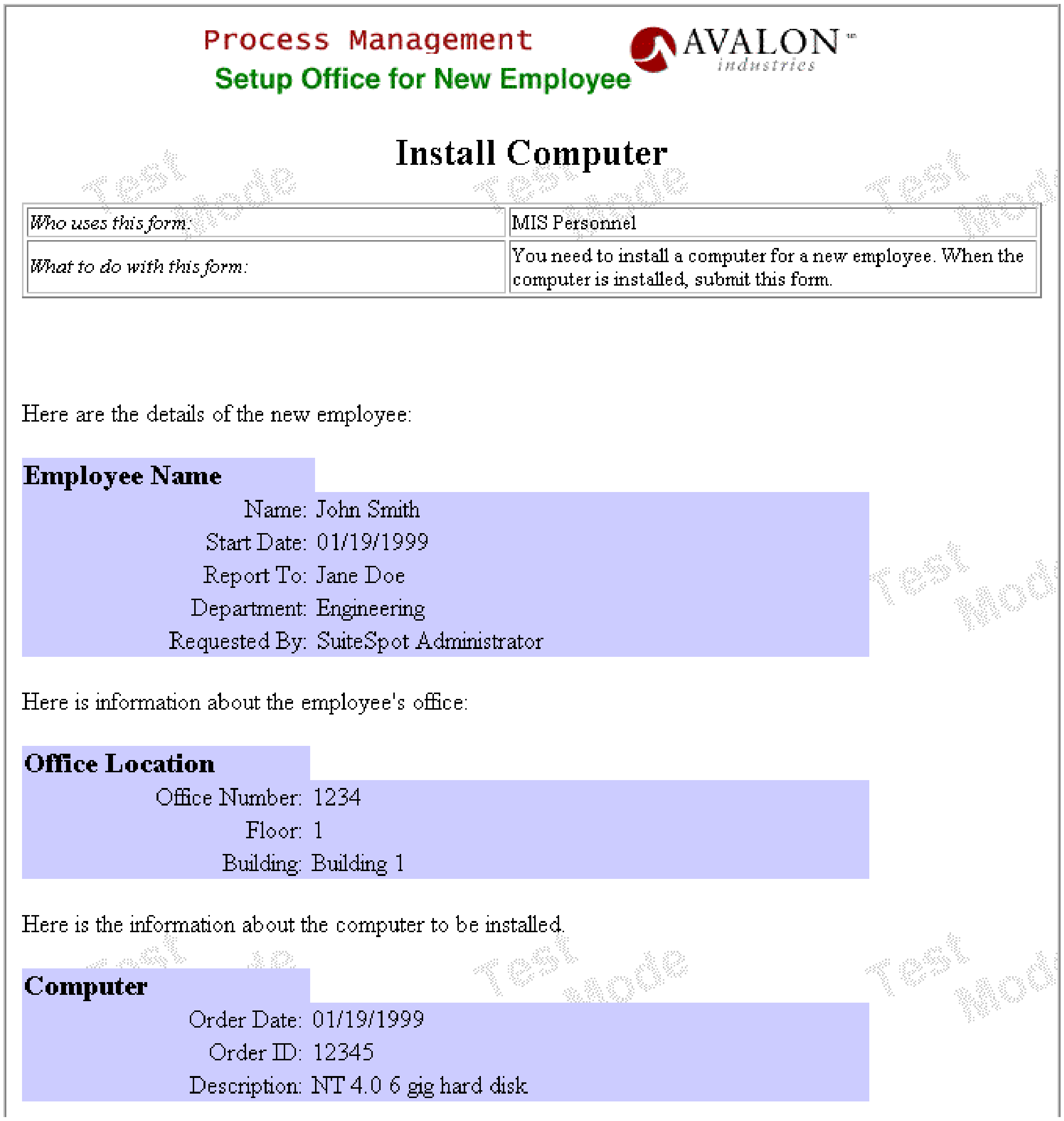
The next step is for the MIS department to install the computer. The process map for this step is shown in Figure 15-13:

Figure 15-13 The Install Computer work item



This work item uses the `installComp.html` form, as shown in Figure 15-14. This form does not set any data field values. Its purpose is to enable the MIS department to specify when the computer is installed so that the office setup process can progress to the next step.

Figure 15-14 The installComp.html form



Install Computer

Who uses this form:	MIS Personnel
What to do with this form:	You need to install a computer for a new employee. When the computer is installed, submit this form.

Here are the details of the new employee:

Employee Name

Name:	John Smith
Start Date:	01/19/1999
Report To:	Jane Doe
Department:	Engineering
Requested By:	SuiteSpot Administrator

Here is information about the employee's office:

Office Location

Office Number:	1234
Floor:	1
Building:	Building 1

Here is the information about the computer to be installed.

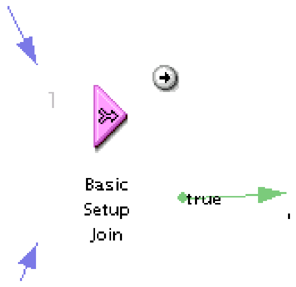
Computer

Order Date:	01/19/1999
Order ID:	12345
Description:	NT 4.0 6 gig hard disk

Basic Setup Join

When the network setup and computer installation is finished, the two parallel branches merge back together into the main branch of the process at the join. The process map for this step is shown in Figure 15-15:

Figure 15-15 The Basic Setup join

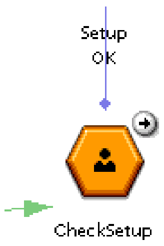


As with a split, the transitions leading away from a join can be conditional. The default condition for each transition is true, and the first transition evaluated that is true is the one followed for a particular process instance.

CheckSetup Work Item

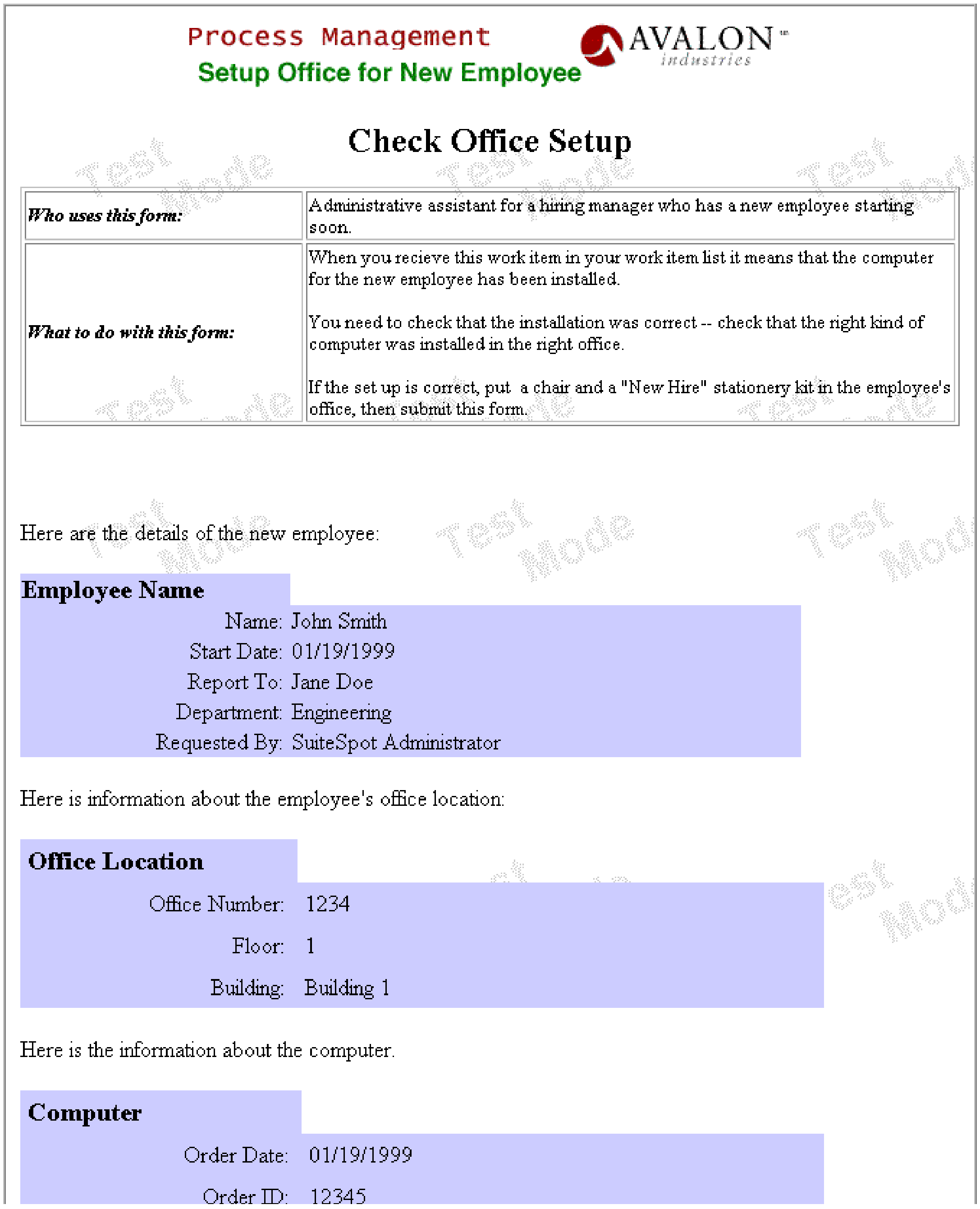
The final step is for the administrative assistant to double-check that the office has been set up properly and has a chair and new package. The process map for this step is shown in Figure 15-16:

Figure 15-16 The CheckSetup work item



This work item has the `checkSetup.html` form, as shown in Figure 15-17.

Figure 15-17 The checkSetup.html form



Setup Office for New Employee

Check Office Setup

Who uses this form:	Administrative assistant for a hiring manager who has a new employee starting soon.
What to do with this form:	When you recieve this work item in your work item list it means that the computer for the new employee has been installed. You need to check that the installation was correct -- check that the right kind of computer was installed in the right office. If the set up is correct, put a chair and a "New Hire" stationery kit in the employee's office, then submit this form.

Here are the details of the new employee:

Employee Name

Name:	John Smith
Start Date:	01/19/1999
Report To:	Jane Doe
Department:	Engineering
Requested By:	SuiteSpot Administrator

Here is information about the employee's office location:

Office Location

Office Number:	1234
Floor:	1
Building:	Building 1

Here is the information about the computer.

Computer

Order Date:	01/19/1999
Order ID:	12345

This work item has a completion script, `verifySetup`, that ensures that the administrative assistant has checked "yes" for the following entries:

Chair in office? This is the `dfChairPresent` radio button data field.

New Hire Stationary Packet on desk? This is the `dfNewHirePacket` radio button data field.

Computer starts correctly? This is the `dfComputerOK` radio button data field.

Setup Complete Exit Point

This application has one exit point, which is reached after the administrative assistant successfully submits the Check Setup form. The exit point is shown in Figure 15-18:

Figure 15-18 The Setup Complete exit point

This exit point has a notification that sends mail to the process instance creator telling them that the process instance has finished. The body of the notification is constructed by the `buildExitNotification` toolkit script.

The Office Setup Groups

This application contains the following application groups:

Purchasing The purchasing department performs the Order Computer work item.

Admin Assistant The administrative assistant performs the Assign Office and Specify Computer work item, as well as the CheckSetup work item.

MIS Dept The MIS department performs the Set Up Phone, Install Network Connection and Install Computer work items.

admin This is the general administrative group.

To make it easier to install and use the sample application in "test" mode, a single `admin` user has been put into each of these groups. If you want to add additional users or change the user, see "Configuring the Office Setup Application" on page 324 for more information.

Data Dictionary

The data dictionary has the following data fields:

dfBldg The building where the new employee's office is located. This is set by the administrative assistant in the Assign Office and Specify Computer work item.

dfChairPresent This is a radio button with choices "yes" and "no" to indicate whether or not the administrative assistant has checked that a chair is present in the new employee's office. This value is set by the administrative assistant in the Check Setup work item.

dfCompOrderDate The date on which the new computer was ordered. This value is set automatically by a client-side script in the Order Computer work item.

dfCompOrderID The order number for the new computer. This value is set by the purchasing department in the Order Computer work item.

dfComputer A description of the computer to order for the new employee. This is set by the administrative assistant in the Assign Office and Specify Computer work item.

dfComputerOK This is a radio button with choices "yes" and "no" to indicate whether or not the administrative assistant has checked that the computer in the new employee's office starts up. This value is set by the administrative assistant in the Check Setup work item. (Although the MIS personnel who installs the computer would be expected to check the installation, the administrative assistant also starts it once to do a double check.)

dfCubeNo The new employee's office number. This is set by the administrative assistant in the Assign Office and Specify Computer work item.

dfDeptName The department that the new employee will be joining. This is set by the hiring manager in the entry point.

dfEmpName The new employee's name. This is set by the hiring manager in the entry point.

dfFloor The floor where the new employee's office is located. This is set by the administrative assistant in the Assign Office and Specify Computer work item.

dfMgr The new employee's manager. This is set by the hiring manager in the entry point.

dfNetworkAddress The IP address for the new employee's network connection. This is set by the MIS department in the Install Network Connection work item.

dfNewHirePacket This is a radio button with choices "yes" and "no" to indicate whether or not the administrative assistant has put a new hire stationery packet in the new employee's office. This value is set by the administrative assistant in the Check Setup work item.

dfPhone The phone number for the new employee. This is set by the MIS department in the Set Up Phone work item.

dfRequesterName The name of the person requesting the office setup. This would usually be the same as the new employee's hiring manager. This is set by the hiring manager in the entry point. This data field would be useful to keep track of the requestor in cases where someone other than the hiring manager is starting the process.

dfStartDate The start date for the new employee. This is set by the hiring manager in the entry point.

Form Dictionary

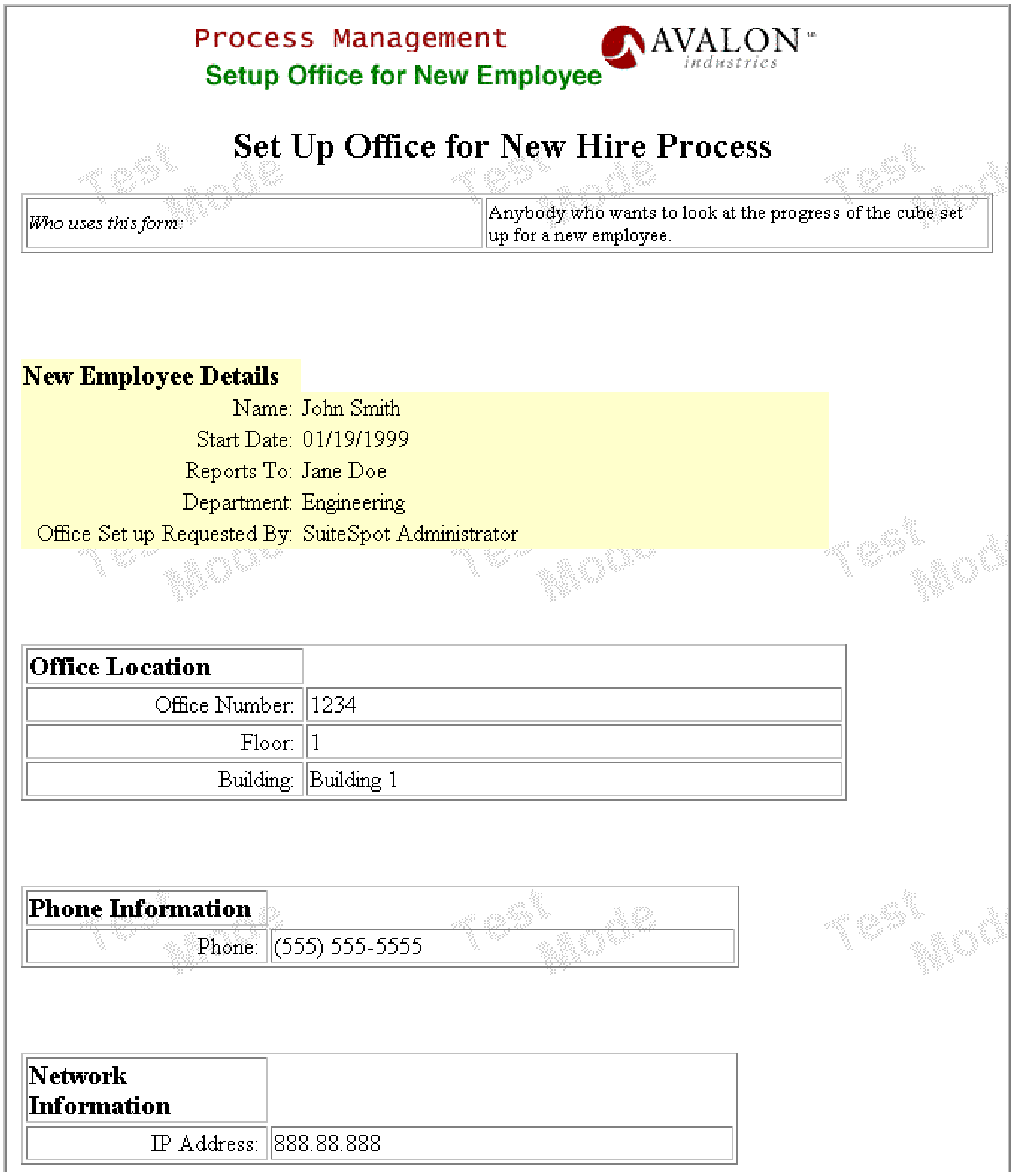
The application uses the following forms:

- assignCube.html
- checkSetup.html
- orderComputer.html
- installComp.html
- prepareForNH.html
- setupNetwork.html
- setupPhone.html
- status.html

All of these forms except the `status.html` form are described in "Office Setup Walkthrough" on page 299.

The `status.html` form, shown in Figure 15-19, is used to show the status of the office setup to interested parties. It shows the data gathered on the other forms in the areas of new employee details, office location, phone information, network information, and computer. It is available to members of "all" beginning with the Assign Office and Specify Computer step and throughout the rest of the process.

Figure 15-19 The status.html form



Set Up Office for New Hire Process

Who uses this form:	Anybody who wants to look at the progress of the cube set up for a new employee.

New Employee Details

Name:	John Smith
Start Date:	01/19/1999
Reports To:	Jane Doe
Department:	Engineering
Office Set up Requested By:	SuiteSpot Administrator

Office Location	
Office Number:	1234
Floor:	1
Building:	Building 1

Phone Information	
Phone:	(555) 555-5555

Network Information	
IP Address:	888.88.888

Script Dictionary

The process uses customized scripts of the following kinds:

- Completion Scripts
- Toolkit Scripts

The process also uses one client-side script to automatically set the order date when a new computer is ordered.

- Embedded Client-Side Script

Completion Scripts

This process uses two completion scripts: `setRequesterField` and `verifySetup`.

- setRequesterField
- verifySetup

setRequesterField

This runs as a completion script on the entry point. It checks the Directory Server for the person that initiated the request, and puts their common name as the value of the `dfRequester` field. If the creator is not found in the Directory Server, the `dfRequester` field is not set. The purpose of setting this field is to store the process instance creator in a data field.

The code for this script is as follows:

```
function setRequesterField(){
  // Get the process instance.
  var pi = getProcessInstance();

  // Get the distinguished name of the creator of the
  // process instance from the Directory Server.
  var creator = pi.getCreatorUser();

  // If we can’t find the creator in the Directory Server, log
  // an error, but it’s not a show-stopping error so keep going.
  if (creator == null || creator == "undefined") {
    logErrorMsg("creator can not be located");
  }

  // If we found the creator in the Directory Server
  // put their common name in the dfRequester data field.
  else {
    var creatorName = creator.cn;
    pi.setData("dfRequesterName", creatorName);
  }

  // Return true so that the activity succeeds
  // and the process continues.
return true;
}
```

verifySetup

This runs as a completion script on the Check Setup work item. This script checks that the `dfChairHere`, `dfComputerOK`, and `dfNewHirePacket` radio button data fields are all set to “yes.” If any of them are not set to “yes,” the completion script returns false, which means the work item cannot be completed.

The plan is that the administrative assistant checks these buttons after checking that the new employee's office has a chair, the computer starts up, and a new hire stationery packet is on the desk.

The code for this script is as follows:

```
function verifySetup(){
var pi = getProcessInstance();

// Return false if any of the dfChairPresent, dfNewHirePacket
// or dfComputerOK data fields are not set to yes.
if ((pi.getData("dfChairPresent") != "yes") ||
    (pi.getData("dfNewHirePacket") != "yes") ||
    (pi.getData("dfComputerOK") != "yes"))

{
// Tell the user why the form could not be submitted.
setErrorMessage("You must ensure that the computer is OK, " +
+ "and there is a chair and new hire packet in the office.");
return false;

}
// If everything’s OK, return true.
return true;
}
```

Toolkit Scripts

There is one toolkit script, `buildExitNotification`.

buildExitNotification

This scripts builds a string to use as the body of an email that gets sent by the exit point to the process instance creator.

This email informs the process instance creator that the process instance has terminated.

The code is as follows:

```
function buildExitNotification(){
  // Get the process instance.
  var pi = getProcessInstance();

  // Get the current work item.
  var wi = getWorkItem();
  var nodeCN = wi.getCurrentActivityCN();

  // Construct a string to use as an email body such as:
  // "The office setup request you initiated on 10/10/1998 for
  // new hire Nikki Beckwell has finished at the Setup Complete
  // exit point."

  var body = "The office setup request you initiated on "
       + pi.getCreationDate()
       + " for new hire "
       + pi.getData("dfEmpName")
       + " has finished at the "
       + nodeCN
       + " exit point.";
return body;
}
```

Embedded Client-Side Script

The `orderComputer.html` form uses a client-side script to automatically set the date on which the computer is ordered. The purchasing department can override this date.

This script makes use of the fact that every data field shown in the form has a corresponding form element of the same name. Thus by setting the value of the `dfCompOrderDate` form element, we are effectively setting the value of the `dfCompOrderDate` data field.

```
function myDate ( ) {
  var today = new Date();

  // getYear() is zeroed around 1900, so add 1900 to the date.
  var currYear = today.getYear();
  currYear += 1900;

  // Jan is 0, Feb is 1 etc... so add 1 to the month.
  var dateString = (today.getMonth() + 1) + "/" +
      today.getDate() + "/" + currYear;

return dateString;
}

document.forms[0].dfCompOrderDate.value = myDate();
```

Customizing the Appearance of the Forms

The forms used in this sample application were edited in an external HTML editing tool to fine-tune their appearance.

All HTML forms link to a single style sheet, `mystyles.html`, that defines a class that is used to define the purple border for the paragraph at the bottom of each form.

The html pages for forms appear at the top level of the subdirectory of Applications that contains your application.

To edit these forms, use an HTML editor that meets the following requirements:

- The editor must ignore all tags and attributes it does not understand, because the data fields are implemented as customized SERVER tags. Some HTML authoring tools will delete tags and attributes that are not standard HTML.
- The editor must leave the following tags upper case: HEAD, BODY, HTML, and SERVER.

HTML editors that meet these requirements include Netscape Communicator (versions 4.5 or later) and Microsoft Internet Explorer (versions 4.0 and later).

If you edit the forms in an external editor, be sure not to modify any of the customized SERVER tags, or the data fields on the forms will not work.

If you edit a web page for a form while it is also open in Process Builder, be sure to click on the form again in Process Builder before saving the application. When you click on a form that has been edited externally, Process Builder asks if you want to load the modified form with the latest changes.

Note that the web page for the form only contain a portion of the information the end-user sees on the form. The Process Engine inserts the banner, buttons, and comment areas of the form, as well as the <FORM> tag itself.

For more information on using an external editor with Process Builder, see "Modifying Forms" on page 176.

Configuring the Office Setup Application

Before you can deploy the Office Setup application, you must make sure your environment meets the following requirements:

- Set the corporate directory as described in "Setting Your Corporate Directory" on page 91.
- Make sure that at least one valid user is present in each of the following groups: Purchasing, MIS Dept, admin, and Admin Assistant.

Typically, you don't need to add users to the above groups, because an `admin` user is added to each of those groups by default. However, you must add users if:

- You do not have an `admin` user in your corporate database.
- You would like to use someone other than the `admin` user or in addition to the `admin` user.

➤ **To add users before deploying the Office Setup application**

1. Make sure that you have defined a corporate directory for the application. For more information, see "Applications and the Corporate Directory" on page 90.
2. In Process Builder, open the application tree view.
3. In the Groups and Roles folder, right-click the group name you want to change, and then choose Properties.

4. Using the Browse option, find the users you want to add, and drag them to the List of users. Or using the Search option, highlight the users and click Add.
5. Close the dialog box.

After you have saved the application, you are ready to deploy it.

Advanced Office Setup Application

The AdvancedOfficeSetup application extends the OfficeSetup application to include a custom data field and custom activity. For details, see the *Process Manager Programmer's Guide*.

Chapter 16

The Loan Management and Credit History Applications

This chapter describes the Loan Management sample application and its subprocess, the Credit History application. These two sample applications show how to use a subprocess in your applications.

This chapter includes the following sections:

Loan Management Application Overview

The Loan Management application controls the process of approving a loan, from the first request to the final approval.

As part of the process, this application contains a subprocess step (Check Credit History) which launches the subprocess application Credit History. This application checks to see how much credit the person applying for the loan has been extended in the past, and returns the information to the Loan Management application. The Loan Management and Credit History applications are specifically designed to show how subprocesses work.

Note that the names for the Check Credit History subprocess and Credit History application are slightly different to distinguish the two. Check Credit History refers to the subprocess icon on the Loan Management process map and to the subprocess item's properties. Credit History refers to the actual application that is called by Loan Management as a subprocess.

Credit History Application Overview

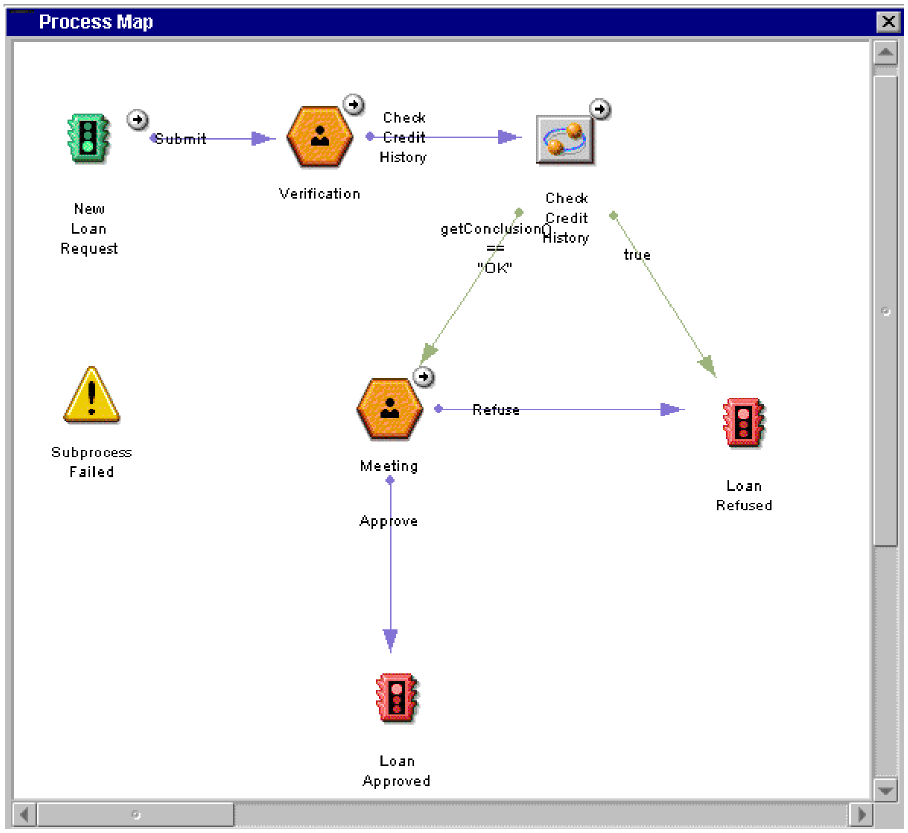
The Credit History application can be run either as a subprocess of the Loan Management application or as a stand-alone application. It shows the process of checking a customer's previous credit. In the real world, that check would probably be done against a database of customer history, but for the purpose of this sample application the amount is hardcoded into the application.

Though the entry point for the Credit History application appears on the process map, it is not included in the walkthrough portion of this chapter. The entry point is only seen by a user if the user runs the Credit History application as a stand-alone application, instead of as a subprocess. Because the walkthrough shows the Credit History application called as a subprocess, the entry point is handled automatically when the Loan Management application starts the subprocess.

Loan Management Process Map

Figure 16-1 shows the process map for the Loan Management application:

Figure 16-1 The Loan Management process map

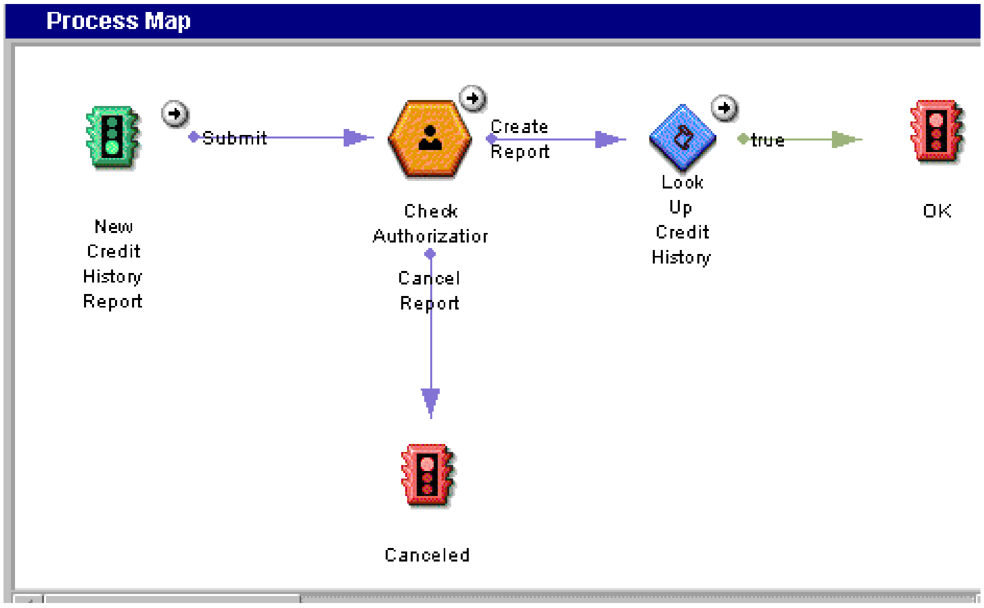


Credit History Process Map

Figure 16-2 shows the process map for the Credit History application.

If run as a stand-alone application, the user starts at the New Credit History Report entry point. If run as a subprocess, the New Credit History Report Entry point happens automatically when this application is called from the Loan Management application.

Figure 16-2 The Credit History process map



Loan Management and Credit History Walkthrough

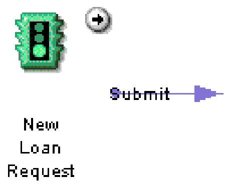
This section walks through the complete process, including both the Loan Management and Credit History applications, step by step. The complete process contains the following entry points, work items (activities), subprocesses, and exit points:

- "New Loan Request Entry Point" on page 331
- "Verification Work Item" on page 333
- "Check Credit History Subprocess" on page 333
- "Credit History Application as a Subprocess" on page 337
- "Check Authorization Work Item" on page 338
- "LookUp Credit History Automated Activity" on page 339
- "Canceled Exit Point" on page 340
- "OK Exit Point" on page 340
- "Meeting Work Item (Parent Process)" on page 342
- "Loan Refused Exit Point" on page 344
- "Loan Approved Exit Point" on page 344

New Loan Request Entry Point

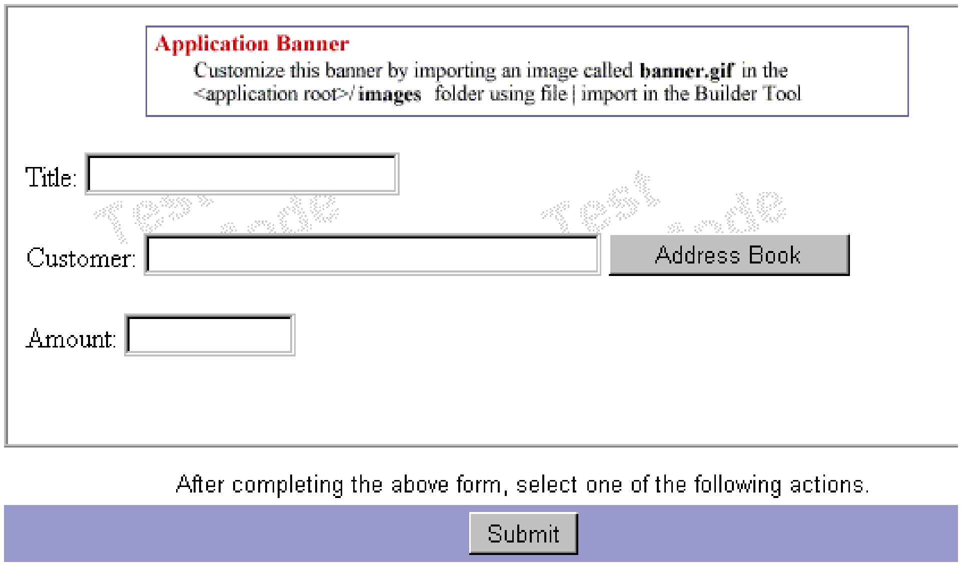
The Loan Management process starts when a user puts in a new loan request. This user is someone at a bank who is responsible for helping customers with their loan applications.

Figure 16-3 The New Loan Request entry point



To start a new loan request, the person at the bank accesses the `creation.html` form in Process Express, as shown in Figure 16-4.

Figure 16-4 The creation.html form



Application Banner
Customize this banner by importing an image called **banner.gif** in the <application root>/**images** folder using file | import in the Builder Tool

Title:
Customer:
Address Book
Amount:
After completing the above form, select one of the following actions.
Submit

This form sets the values of the following data fields:

title The title of the process instance. Since it appears on the work list, this field needs a value that is meaningful to the end users, for example, the kind of loan requested.

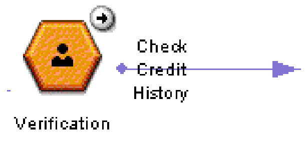
customer The name of the customer who is requesting the loan. This is a user picker widget. The customer name is used for the field role Customer. This user needs to be in the corporate directory.

amount The amount of money the loan is for.

Verification Work Item

The next step is for the creator of the loan application to verify the information entered to make sure that it is correct. The process map for this step is shown in Figure 16-5.

Figure 16-5 The Verification work item

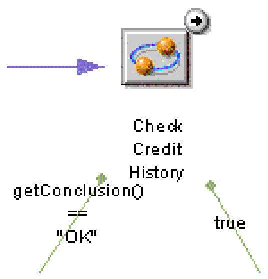


This step uses the same form as the entry point, `creation.html`, shown in Figure 16-4. The only additional field is the “comments” text area, where the creator can enter comments that will appear in the details and history. The comments section appears because the Allow to Add Comments property for the Verification activity is set to true.

Check Credit History Subprocess

At this step the Loan Management application calls a subprocess, the Credit History application. The process map for this step is shown in Figure 16-6:

Figure 16-6 The Check Credit History subprocess



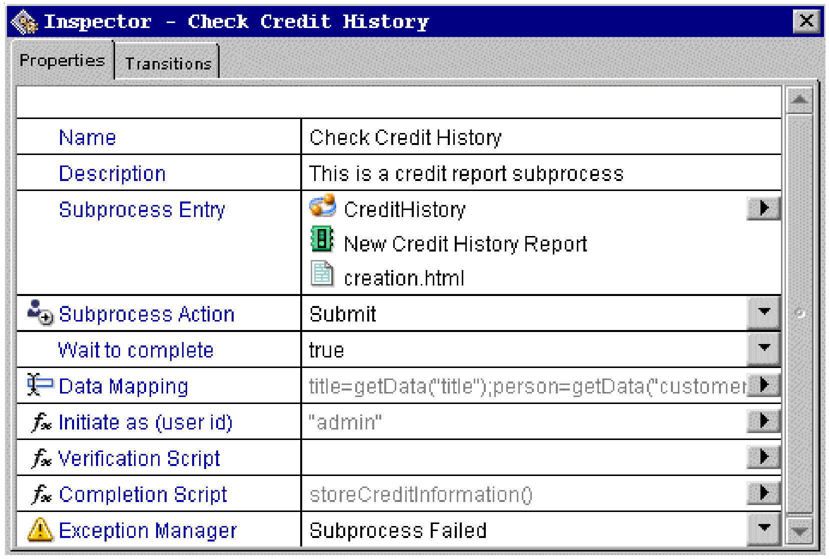
This subprocess checks the credit history of the customer. If the subprocess terminates in the OK exit point, the loan request moves forward to the next step. If the Credit History application does not end at the OK exit point, the loan is denied.

Check Credit History Subprocess Properties

The Check Credit History subprocess in Loan Management has the properties as shown in Figure 16-7. (You can view properties by right-clicking the subprocess item in the process map and choosing Properties.)

The end user does not see these properties, but they govern how the Loan Management application starts the Credit History application as a subprocess.

Figure 16-7 Check Credit History subprocess properties dialog box



Name	Check Credit History
Description	This is a credit report subprocess
Subprocess Entry	CreditHistory New Credit History Report creation.html
Subprocess Action	Submit
Wait to complete	true
Data Mapping	title=getData("title");person=getData("customer
Initiate as (user id)	"admin"
Verification Script	
Completion Script	storeCreditInformation()
Exception Manager	Subprocess Failed

The Subprocess Entry shows that the subprocess application is CreditHistory, that the entry point accessed for this application is the New Credit History Report entry point, and that the form at this entry point is `creation.html`. A subprocess must use an entry point that only has one form. The fields on `creation.html` that would ordinarily be filled in by the user at the entry point are filled in automatically using data mapping. For more information, see "Data Mapping" on page 335.

The Subprocess Action is Submit, which is the transition that leads from the entry point to the first activity on the Credit History process map.

The "Initiate as" user ID is the ID of the user who initiates a subprocess. In many cases this field would use a script to determine the user by process instance. However, in this sample it is hardcoded to the static value of the `admin` user. If you are not using the `admin` user, or want a different user, you can replace this value. For information about revising the defined users, see "Configuring the Credit History Application" on page 350.

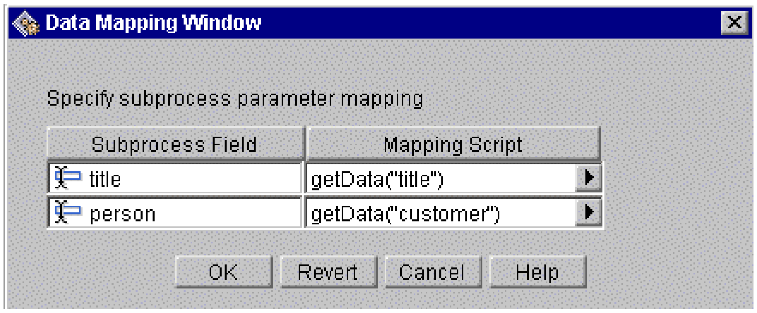
The Data Mapping, Completion Script, and Exception Manager properties are covered in the following sections.

The Transitions tab shows the order in which to evaluate the transitions out of the subprocess. First, the application evaluates the transition named `getConclusion() == "OK"`. This transition checks if the subprocess ended at the OK exit point. If that is true, the process continues to the Meeting work item. If the subprocess did not end at the OK exit point, the next transition is evaluated. Because the next transition is `true`, it acts as an "else" statement. If the subprocess ended anywhere except the OK exit point, the process follows this transition to the Loan Refused exit point.

Data Mapping

In the Data Mapping dialog box, the parent process fields are mapped to the subprocess fields that are required to complete the form at the subprocess's entry point. Figure 16-8 shows the dialog box for this sample application.

Figure 16-8 Data Mapping setting dialog box


Data Mapping Window

Specify subprocess parameter mapping

Subprocess Field	Mapping Script
title	getData("title")
person	getData("customer")

In this sample application, the `creation.html` form (the form at the entry point of the Credit History application) contains two fields that must be filled in: `title`, and `person`.

The `title` field in the subprocess is mapped to the value in the parent process's `title` field. The script `CustomerId()` populates the subprocess data field `person` with the user ID of the customer stored in the parent process's `customer` field. For more information on this script, see "CustomerId Toolkit Script" on page 348.

Completion Script

The completion script `storeCreditInformation` takes the values from the Credit History application and stores them in the appropriate data fields in the Loan Management application. It also determines at what exit point the subprocess ended. For more information, see "storeCreditInformation Completion Script" on page 347.

The Subprocess Failed Exception Manager

If the subprocess fails for some reason, the application calls an exception manager named Subprocess Failed. The tree view shows this exception manager as a yellow triangle. It also appears on the process map, as shown in Figure 16-9, but it is not connected by transitions to any other items.

Figure 16-9 Exception manager

The exception manager is similar to an activity, in that you assign it to someone (usually the administrator), and that person can view a form and take action. For the Loan Management application, the exception manager is assigned to the creator by default, and the assignee views the `create.html` form.

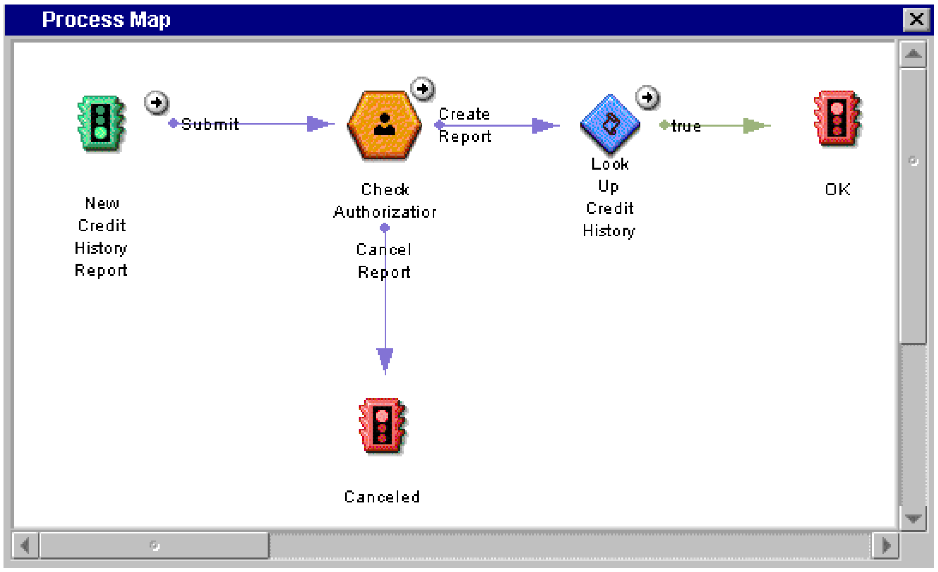
The Subprocess Failed exception appears on a work list only if the subprocess fails. In most cases it won't appear.

Credit History Application as a Subprocess

When the process reaches the Check Credit History step, the Credit History application is started as a subprocess of the Loan Management application.

The process map for the Credit History application is shown in Figure 16-10. The process map displays the steps that take place before returning to the Loan Management parent process.

Figure 16-10 The Credit History process map

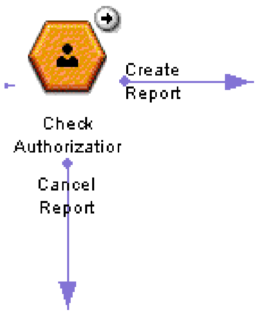


When Loan Management calls Credit History, the New Credit History Report entry point happens automatically. The fields in the form `creation.html` are filled in by data mapping, and the Submit action is performed automatically.

Check Authorization Work Item

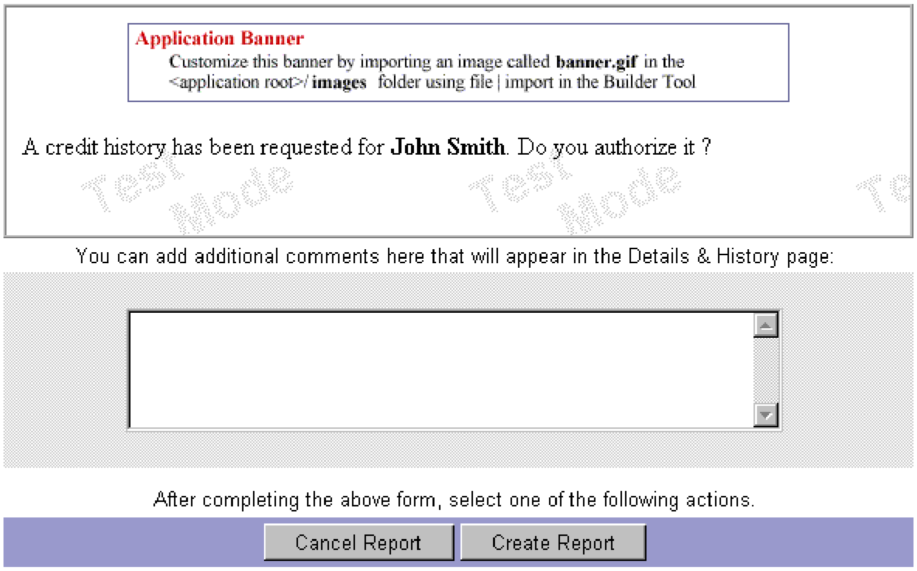
The next step in the process is the first activity in the Credit History application, Check Authorization. The process map for this step is shown in Figure 16-11.

Figure 16-11 The Check Authorization work item



This work item is completed by the person responsible for authorizing credit reports. However, for the purposes of the sample application, this work item is assigned by default to the creator. The form associated with this step is `form.html`. This form contains a read-only field, `person`, which is the field role associated with the customer requesting the loan. This field role is picked up from the parent process, Loan Management. In Figure 16-12, the request has been made for the user's SuiteSpot Administrator.

Figure 16-12 The form.html form



Application Banner

Customize this banner by importing an image called **banner.gif** in the <application root>/**images** folder using file | import in the Builder Tool

A credit history has been requested for **John Smith**. Do you authorize it ?

You can add additional comments here that will appear in the Details & History page:

After completing the above form, select one of the following actions.

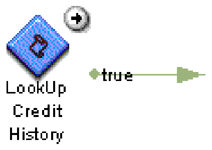
The choices available from this step are Cancel Report and Create Report.

If you choose Cancel Report, the Credit History application continues to the Canceled exit point. If you choose Create Report, the process continues to the automated activity LookUp Credit History.

LookUp Credit History Automated Activity

The next step is the automated activity Look Up Credit History. The process map for this step is shown in Figure 16-13.

Figure 16-13 The Look Up Credit History automated activity



In a real-world situation, this automated activity would look up a customer's credit history in a database. However, for the purpose of this sample application, the values are hardcoded. The value of the data field `credit_history` is set to 10000.95. The Loan Management process uses this amount, along with the date, when the Credit History subprocess returns to the Loan Management application.

After this automated activity runs, the process continues to the OK exit point for the Credit History application.

Canceled Exit Point

If the report is canceled at the Check Authorization step, the process goes to the Canceled exit point, shown in Figure 16-14.

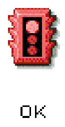
Figure 16-14 The Canceled exit point

This step displays the form `refused.html`. This form contains no fields, just the message that the credit history report is not authorized. From this exit point, the subprocess returns to the Loan Management process and continues to the Loan Refused exit point.

OK Exit Point

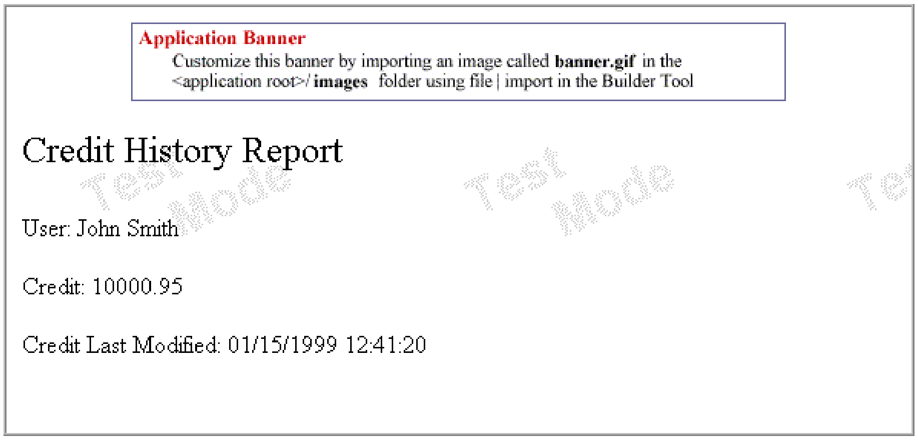
If the request for the report is approved, the process goes to the OK exit point, shown in Figure 16-15.

Figure 16-15 The OK exit point



This step displays the form `completed.html`, shown in Figure 16-16.

Figure 16-16 The completed.html form


Application Banner
Customize this banner by importing an image called **banner.gif** in the <application root>/ **images** folder using file | import in the Builder Tool

Credit History Report

User: John Smith

Credit: 10000.95

Credit Last Modified: 01/15/1999 12:41:20

This form shows the information gathered by the `LookUpCreditHistory` script. It has the following fields:

person The person for whom the credit history is gathered. The same as customer in the Loan Management application.

credit_history The amount of money the person has been loaned in the past.

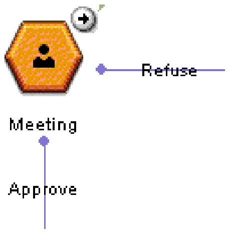
credit_date The date the credit history check was performed.

From this exit point, the subprocess returns to the Loan Management process and continues to the Meeting work item.

Meeting Work Item (Parent Process)

The Check Credit History subprocess has a completion script, `storeCreditInformation`. If this script finds that the subprocess concluded at the OK exit point, the subprocess proceeds to the Meeting step, shown in Figure 16-17.

Figure 16-17 The Meeting work item

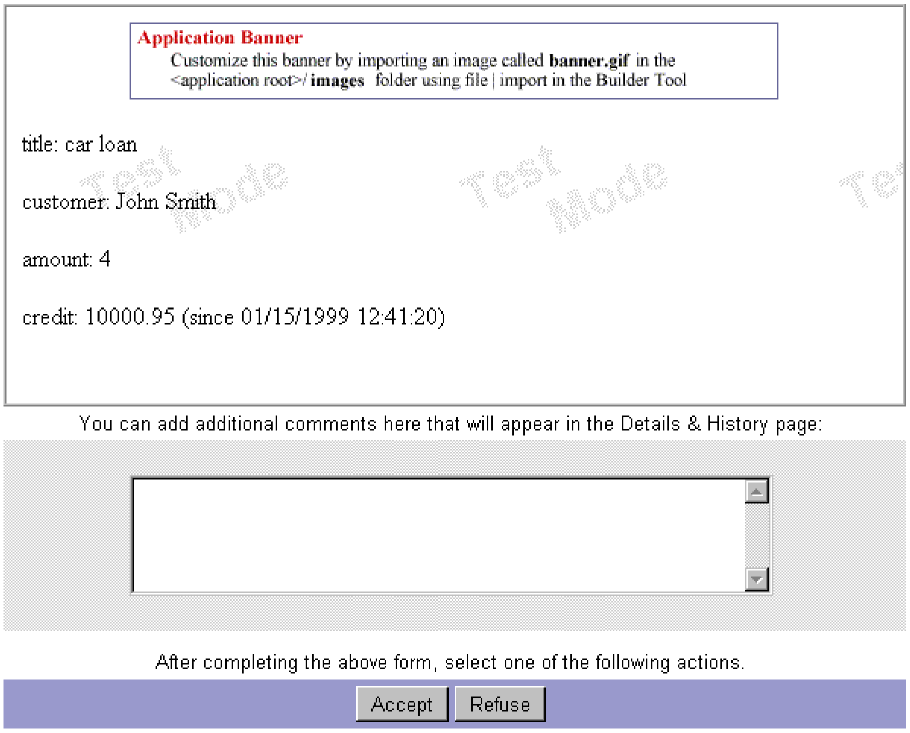


In the Meeting step, the person making the loan meets with the customer to review the loan contract's details. For this sample application, the Meeting activity is assigned by default to the creator of the process instance.

At this point, the customer can either approve or refuse the terms. If the customer approves the terms, the process continues to the Loan Approved exit point. If the customer refuses the terms, the process continues to the Loan Refused exit point.

The form displayed at this step is `summary.html`, shown in Figure 16-18.

Figure 16-18 The summary.html form


Application Banner
Customize this banner by importing an image called **banner.gif** in the <application root>/ **images** folder using file | import in the Builder Tool

title: car loan

customer: John Smith

amount: 4

credit: 10000.95 (since 01/15/1999 12:41:20)

You can add additional comments here that will appear in the Details & History page:

After completing the above form, select one of the following actions.

Based on the results of the meeting, the two actions available to the person making the loan are Accept or Refuse. Clicking Accept sends the process to the Loan Approved exit point, whereas clicking Refuse sends the process to the Loan Refused exit point.

Note that this form is also used throughout the process for the group "all," so that everyone can see a summary of the loan process. This form is also used at the "Subprocess Failed" exception manager.

This form contains the following read-only data fields. They are the data fields from the original loan creation form, along with information gathered in the subprocess:

title The title of the process instance. Since it appears on the work list, this field needs a value that is meaningful to the end users, for example, the kind of loan requested.

customer The name of the customer who is requesting the loan. This is a user picker widget. The customer name is used for the field role Customer. This user needs to be in the corporate directory.

amount The amount of money the loan is for.

credit The amount of credit the customer has had, as determined by the Check Credit History subprocess.

last_modified The date the credit was checked.

Loan Refused Exit Point

If the customer refused the loan terms at the Meeting step, or if the Credit History process ended at the Canceled exit point, the next step is the Loan Refused exit point, shown in Figure 16-19.

Figure 16-19 The Loan Refused exit point

The form for the exit point is `summary.html`, shown in Figure 16-18. However, since the process is at an exit point, the fields are display-only, and no action can be taken.

Loan Approved Exit Point

You reach the Loan Approved exit point, shown in Figure 16-20, if the loan is approved in the Meeting step.

Figure 16-20 The Loan Approved exit point

The form for the exit point is `summary.html`, shown in Figure 16-18. However, since the process is at an exit point, all the fields are display-only, and no action can be taken.

Groups and Roles

The Loan Management application has the following non-default role:

customer A field-based role representing the customer who requested the loan. The data field `customer` is mapped to the `person` data field in the Credit History application, and so has the same value.

The Credit History application has the following non-default groups and roles:

person A field-based role representing the person who requested the loan. The data field `person` is mapped to the `customer` data field in the Loan Management application, and so has the same value.

trusted users An application group that is composed of users trusted by the application to start the application as a subprocess. These user IDs are the "Initiate as (user IDs)" of the parent processes that would call the process as a subprocess. For the Credit History application, this group by default contains one user, "admin," which is the default "Initiate as" user ID for the Loan Management application. You may have changed this value when you configured the Loan Management application. For more information on trusted users, see "The Corporate Group Dialog Box" on page 138.

For more information about configuring the groups and roles for these sample applications, see "Configuring the Loan Management Application" on page 349 or "Configuring the Credit History Application" on page 350.

Data Dictionary

The Loan Management application has the following data fields in its data dictionary:

amount The amount of the loan that the customer is applying for.

credit The amount of credit the customer was last given, based on the credit history.

customer The name of the customer who is requesting the loan. This is a user picker widget. The customer name is used for the field role Customer. This user needs to be in the corporate directory.

last_modified The date the credit history was last modified.

title The title of the process instance. Since it appears on the work list, this field needs a value that is meaningful to the end users, for example, the type of loan requested.

The Credit History application has the following fields in its data dictionary:

credit_date The date the credit report was last updated.

credit_history The amount of credit the customer was last given.

person The user ID of the person whose credit history is requested.

title The title of the process instance. Since it appears on the work list, this field needs a value that is meaningful to the end users, for example, the kind of credit history requested.

Loan Management Script Dictionary

The Loan Management application contains two scripts:

- “storeCreditInformation Completion Script” on page 347
- “CustomerId Toolkit Script” on page 348

storeCreditInformation Completion Script

This script runs after the subprocess is complete. It finds out which exit point the subprocess reached. The script also gets the values of the fields set in the subprocess (`credit_history` and `credit_date`) and sets the values of the data fields in the parent process (`credit` and `last_modified`) to those values. Notice that this script takes advantage of the `getSubProcessInstance` function to get the subprocess instance data and state. The script's code is as follows:

```
function storeCreditInformation(){
  // Get Handle on subprocess instance
  var spi = getSubProcessInstance();

  // Get Handle on process instance
  var pi  = getProcessInstance();

  // getConclusion returns the name of the exit point reached
  // by the subprocess
  switch( getConclusion() ){
    case "OK":
    // The subprocess instance has been successfully completed

      // Read data from the subprocess instance.
      // Please note that "credit_history" and "credit_date"
      // are names of fields defined in the child process.
      var credit_history = spi.getData("credit_history");
      var credit_date    = spi.getData("credit_date");

      // Store data in process instance.
      // Please note that "credit" and "last_modified"
      // are names of fields defined in this process.
      pi.setData("credit",        credit_history);
      pi.setData("last_modified", credit_date);

      // Success
      return true;

    default:
      // No data transfer required
      return true;
  }
}
```

CustomerId Toolkit Script

This script is used during the data mapping from the parent process to the subprocess. It takes the value of the `customer` field in the Loan Management application and finds the associated user ID for that customer. The user ID is then entered into the `person` field in the subprocess application, Credit History.

This script is needed because field roles are entered as user IDs but are stored as distinguished names. Therefore, the value from the `customer` field must be converted from a distinguished name to a user ID before it can be entered into the `person` field.

The script's code is as follows:

```
function CustomerId(){
  // Get a handle on the process instance;
  var pi = getProcessInstance();

  // Get the customer user.
  var cus = pi.getRoleUser( "customer" );

  // Verify that the role has been populated correctly.
  if ( ( cus == "undefined" ) || ( cus == null ) ){
    logErrorMsg("CUSTOMER_NOT_DEFINED");
    return null;


  // return the user ID of the customer
  return cus.uid;
}
```

Credit History Script Dictionary

The Credit History application has one script, an automation script called `LookUpCreditHistory`.

LookUpCreditHistory Automation Script

This script is used by the automated activity LookUp Credit History. This script determines the values for the `credit_history` and `credit_date` fields. In a real-world situation, this data might be contained in a database. However, for the purpose of this sample application, the values are hardcoded into the script.

```
function LookUpCreditHistory(){
  // Get a handle on the Process Instance.
  var pi = getProcessInstance();

  // Get the information about the user whose
  // credit history is requested.
  var user   = pi.getRoleUser("person");
  var userId = user.uid;

  // Based on the userId, determine the
  // credit_history and the credit_date.
  // These values are hardcoded, but we could instead
  // access an external database for more information.
  pi.setData( "credit_history", 10000.95 );
  pi.setData( "credit_date"   , new Date() );

  return true;
}
```

Configuring the Loan Management Application

Before you can deploy the Loan Management application, you must make sure your environment meets the following requirements:

- Set the corporate directory as described in "Setting Your Corporate Directory" on page 91.
- Add a user to the `admin` group.

➤ **To add a user to the `admin` group**

1. In the application tree view, right-click the `admin` group and choose Properties. Or highlight the group and click the Inspector button from the toolbar.

2. Using the Browse option, find the users you want to add, and drag them to the List of users. Or using the Search option, highlight the users and click Add.
3. Close the dialog box.

After you have set these properties, you can deploy the Loan Management application. However, you cannot run the application until you also deploy the application it uses as a subprocess, Credit History.

Configuring the Credit History Application

Before you can deploy the Credit History application, you must make sure your environment meets the following requirements:

- Set the corporate directory as described in "Setting Your Corporate Directory" on page 91.
- Add users in two groups: admin and trusted users. The admin can be anyone, but the default is the `admin` user. The trusted user must be the same user as defined in the Loan Management application for the Credit History subprocess.

➤ **To check this user value**

1. Open the LoanMgmt application.
2. Open the Inspector window for the "Check Credit History" activity.
3. Check the value for the "Initiated As (user id)" property. By default, this is set to `admin`.

➤ **To add a user to the admin or trusted user group**

1. In the application tree view, right-click the group and choose Properties. (You can also double-click the group, or highlight the group and click Inspector from the toolbar.)
2. Using the Browse option, find the users you want to add, and drag them to the List of users. Or using the Search option, highlight the users and click Add.
3. Close the dialog box.

After you have saved the application, you are ready to deploy it.

Chapter 17

The Insurance Claim Processing Application

This chapter describes the Insurance Claim Processing sample application that is shipped with this product. It includes a general walk through of the sample application and describes its functions. You can use the applications as a learning tool to see how applications are designed.

This chapter contains the following sections:

Application Overview

This application automates an insurance claims process from the time a customer enters a claim for automobile damage to the time when the claim is either approved or denied. The application demonstrates how you can implement a number of advanced Process Manager features including the following:

- Expiration for customer forms

  The Application uses expiration setter and handler scripts in order to discard forms that are not accessed for a certain time.

- Accessing a flat file

  The application takes the policy number from the customer and reads a flat file (an XML file with customer records) and parses it to get the policy details record for the customer. If the record is not found the application prompts the user to re-enter the policy number.

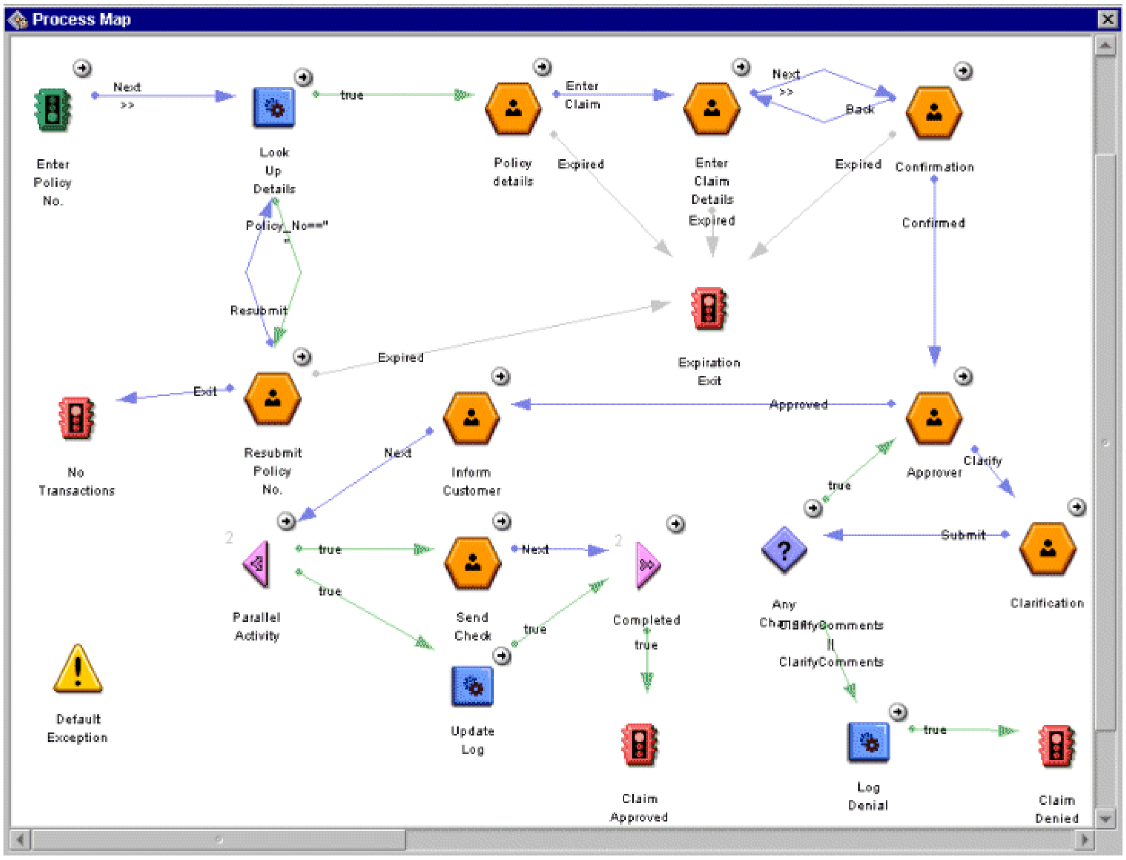
NOTE The application contains all the necessary forms, user roles, and database fields to complete the process.

You must use Netscape Communicator to run this application.

Process Map

Figure 17-1 shows the application's steps:

Figure 17-1 The Insurance Claim Process Map



The process map shows the following steps.

Entry Point

The process map contains one entry point where the user enters their policy number. Immediately following the entry point, a custom activity uses the policy number to retrieve the customer's record from the flat file database.xml.

Activities

This process has eight manual activities:

- **Policy Details.** Displays the customer's personal information and policy details
- **Re-Enter Policy No.** Prompts the user to re-enter the policy number if it was not found in the flat file.
- **Enter Claim Details.** Prompts the user to enter the claim details.
- **Confirmation.** Displays the customer's personal information, policy details and details of the claim. The customer can confirm the details or go back to the previous form to correct the details.
- **Approver.** Allows an insurance adjuster to approve the insurance claim if everything is correct or to request clarification of the claim
- **Clarification.** Allows an insurance clerk to enter changes to the claim if applicable after talking to the customer to clarify the claim.
- **Inform Customer.** Informs the customer that the claim was approved.
- **Send Check.** Informs the customer that the check was sent.

Custom Activities

There are three custom activities, which perform different functions.

Look Up Details

This custom activity is the most important of the three custom activities. It takes the policy number, entered by the customer at the entry point, as the INPUT. It then opens the flat file (database.xml) and parses the file to get the matching policy number as entered by the customer.

If it finds a matching policy number, it fetches the records and sends all the details to the OUTPUT of the custom activity. All this is done in the perform() function of the custom activity Java file. The record is mapped to the variables in Process Builder and hence when we view the next form we can see all the details.

If it does not find a matching policy number, it prompts the customer to re-enter the policy number until it finds the matching policy number in the flat file.

It maintains a log file (Policy.log) which is created if not present. This log file has the information about all the access to the policy and the results.

NOTE The flat file (database.xml) must be in the current user directory in order to make the application work.

Update Log

This custom activity updates the Policy.log file when the claim is approved. It is a parallel activity.

Log Denial

This custom activity updates the Policy.log whenever the claim is denied.

Decision Point

After the Clarify activity, there is a decision point. If no changes have been made to the claim details, the process returns to the Approver activity so that the insurance adjuster can re-examine the claim. If no changes have been made, however, the process moves down the denial path. This is because if there are no changes then the claim is not correct and hence denied.

Exit Points

The process has four exit points:

- **Claim Approved.** A claim is approved.
- **Claim Denied.** A claim is denied.
- **Expiration Exit.** Whenever an Expiration handler gets activated it destroys the process instance.
- **No Transactions.** When there is no access to the policy details such as when the user enters the wrong policy number and then presses the exit button in the form that asks them to re-enter the number.

Application Walkthrough

This walkthrough takes you through the process described by the application, describing the portions of the application involved in each step.

The Entry Point (Enter Policy No.)

Users (customers) initiate an insurance claim at the Enter Policy No. entry point. When they initiate the application in Process Express, they see the Entry.html form. On this form customer enters the policy number for which they are claiming the insurance.

Custom Activity (Look Up Details)

When users have filled out the form, they click Next to perform the function of the class file associated in the custom activity - Look Up Details. The function of this custom activity is to open database.xml (which is the flat file), parse it, and look for a matching policy number.

A main function of this application is to log all the activities carried out whenever a new process is initiated. This helps the insurance company personnel check all insurance claims entered through this application. At this custom activity, the application adds the customer name and a randomly generated claim ID to a file called Policy.log.

Activity (Policy Details)

If the custom activity finds a matching Policy number, it displays the policy details on the form `Details.html`. This activity has two scripts associated with it:

- "Script at Policy Details Activity"
- "Completion Script at Policy Details Activity"

Activity (Re-enter Policy No.)

If the custom activity does not find the policy number in the flat file, it will prompt the customer to re-enter the policy number until it finds a policy record. It prompts the customer by displaying a form, called Resubmit.html.

Activity (Enter Claim Details)

The form at this activity, `EnterClaim.html`, allows the customer to enter the claim. It has predefined claim such as vehicle damage, window damage etc. The customer clicks the appropriate check box to select the type of claim and describes the claim in detail in the text area provided.

This activity also has two scripts associated with it: "Script at Enter Claim Details Activity" on page 370 and "Completion Script at Enter Claim Details Activity" on page 372.

Activity (Confirmation)

After entering all the details in the EnterClaim.html form, the form at this activity Confirmation.html shows all the details associated with the customer and the policy, including the claim details which the customer had entered in the previous form. This form prompts the customer to check all the details displayed and then continue if correct.

If the customer feels that there is a error in the claim details a 'BACK' button takes the customer back to the Enter Claim Details activity, where the customer can enter the corrects claim details and continue.

This activity also has an Script at Confirmation associated with it.

Activity (Approver)

After the confirmation from the customer, the process continues to the next step, approving the claim. This activity is carried out by the insurance company employee who has the rights to approve the claim. The form Approver.html displays all the information shown previously in the Confirmation.html, the customer personal information, the policy details, and the claim. In addition to this information, the form contains a text area for comments from the approver.

The approver can approve the claim at this point or, if there are some comments or doubts about the claim or the customer, the approver can send it on for clarification.

Activity (Clarification)

If the approver in the previous activity has some doubts about the claim or the customer, the approver can send it for clarification. The form, Clarify.html, has the same contents as Approver.html, and an additional field for the clarification comments. The clarifier talks with the customer and looks for any changes in the policy. If there are any changes he writes them in the Clarification Comments text area and submits it to the approver.

If there are no changes in the policy, then the approver's comments hold good and there is something wrong with the claim. In this case, the claim is denied.

Activity (Inform Customer)

If the approver approves the claim, this activity displays a confirmation form (InfCustomer.html) that the claim with Customer Name, Policy No. and Address to which the claim check will be sent is approved.

Parallel Activities

After the Inform Customer activity, two different tasks are carried out in parallel: Sent Check and Update Log.

Activity (Send Check)

The first part of the parallel activity informs the insurance claim personnel that the check for the approved claim was sent. This is displayed in the SendCheck.html form.

Custom Activity (Update Log)

The other part of the parallel activity is a custom activity - Update log. This custom activity updates the log file, which was appended to when the process was initiated. This custom activity writes the claim ID, which was randomly generated during the process initiation, and the claim, which was approved, in the log file.

Groups and Roles

The following groups and roles are defined for this application:

- all
- creator
- admin
- trusted users

These are the default groups and roles that are created for every application. For demonstration purposes, all activities in this application are assigned to the creator of the process instance. In order for a group to be able to search, the Allow Search option must be checked in the Inspector window for that group.

Group and Role Priorities

The group and role priorities determine the order in which the application processes groups and roles to determine what forms the users see. Because a user can belong to more than one group or role in the same application, the application needs the builder to specify which group or role to evaluate first. The evaluation order you set in the Groups and Roles properties window is the order used in the Form Access window.

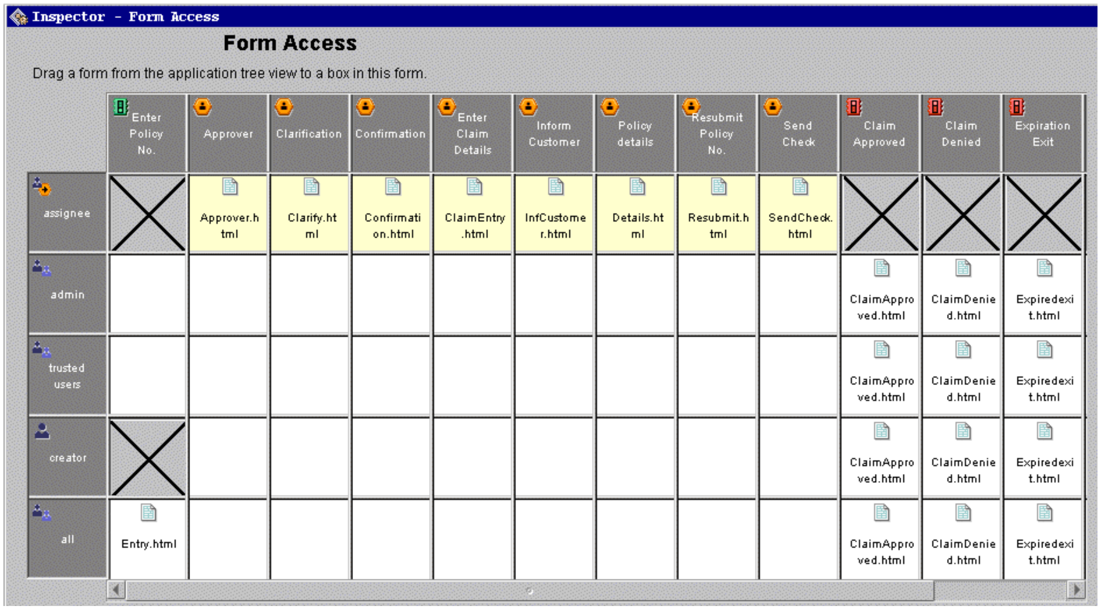
The assignee role is always at the top of the list and the “all” group is always at the bottom. You cannot reprioritize these two.

➤ **To see the group and role priorities**

1. In the application tree view, double-click the Groups & Roles folder, or highlight the folder and click Inspect.
2. In the Inspector Window, click “evaluation order” if it's not already selected. The groups and roles are listed in the following order:
   - assignee
   - creator
   - admin
   - all

The application first checks to see if a user is the assignee of a particular activity. If so, it finds the appropriate form to display to the assignee. If the user is not the assignee, the application checks down the list, and when it finds a role or group the user belongs to, it displays the appropriate form as configured in the Form Access window. Figure 17-2 shows a portion of this window:

Figure 17-2 Form Access Window



Inspector - Form Access

Form Access

Drag a form from the application tree view to a box in this form.

	Enter Policy No.	Approver	Clarification	Confirmation	Enter Claim Details	Inform Customer	Policy details	Resubmit Policy No.	Send Check	Claim Approved	Claim Denied	Expiration Exit
assignee	✗	Approver.html	Clarify.html	Confirmation.html	ClaimEntry.html	InfCustomer.html	Details.html	Resubmit.html	SendCheck.html	✗	✗	✗
admin										ClaimApproved.html	ClaimDenied.html	Expiredexit.html
trusted users										ClaimApproved.html	ClaimDenied.html	Expiredexit.html
creator	✗									ClaimApproved.html	ClaimDenied.html	Expiredexit.html
all	Entry.html									ClaimApproved.html	ClaimDenied.html	Expiredexit.html

Data Dictionary

The data dictionary contains all the data fields that are used by the application and stored in the database. To see the properties for these fields, in the application tree view, inside the Data Dictionary folder, double-click the name of the field.

Fields are defined to have a class ID, which in turn determine the properties the field has. When you add fields, you can either add them with predefined class IDs, or you can create your own class IDs.

Table 17-1 shows the Data Dictionary for the Insurance Claim Processing application:

Table 17-1 Data dictionary for Insurance Claim Processing application

Data Field	**Purpose**
Customer_Name	Stores the customer name
Customer_Address	Stores the customer address.
Customer_Email	Stores the customer email, so that the email can be sent to the customer whenever a claim is approved, denied or a process is expired.
Customer_PhoneNo	Stores the customer phone number, which can be needed in case you need to contact the customer for clarifications.
Policy_No	Stores the policy number of the customer.
Start_Date	Stores the policy start date.
End_Date	Stores the policy end date.
Policy_Amt	Stores the policy amount
Deductible	Stores the policy deductible amount.
Vehicle_Model	Stores the vehicle model
Vehicle_IDN	Stores the identification number for the vehicle
AppComments	Stores the approver’s comments
CalrifyComments	Stores the clarifier’s comments.
Claim	Stores the customer’s claim
claimid	Stores a randomly generated claim ID
Window_damage	checkbox for predefined claim - window damage
Body_damage	checkbox for predefined claim - body damage
Engine_damage	checkbox for predefined claim - engine damage
Accessories_damage	checkbox for predefined claim - accessories damage

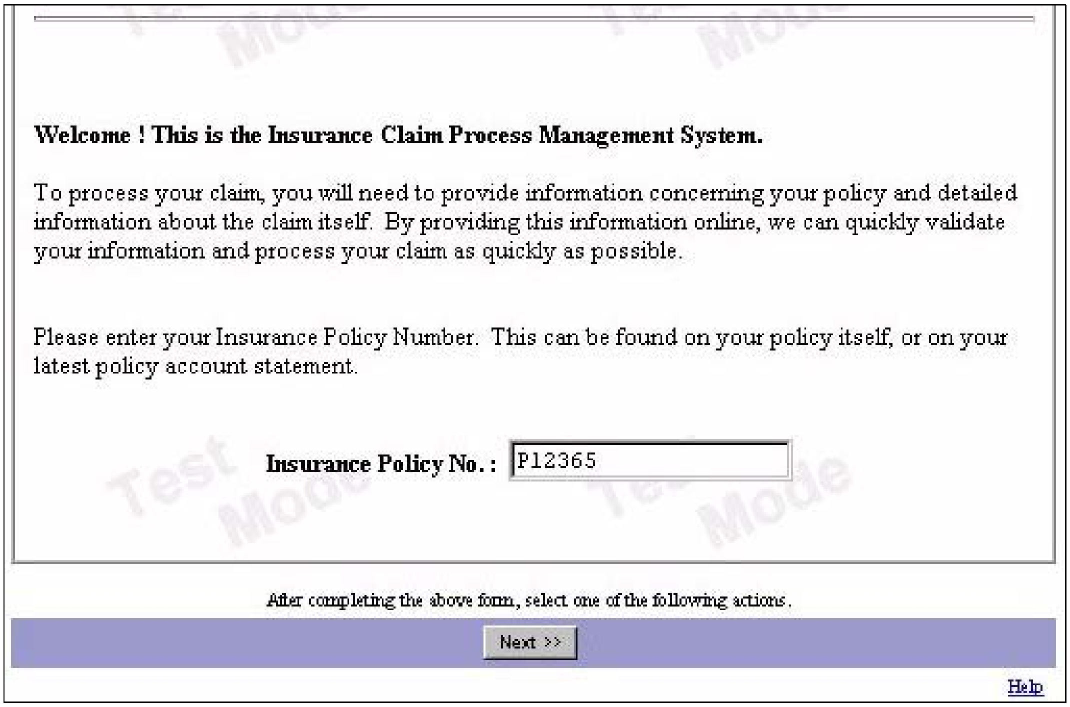
Forms

The following sections describe the forms included in the Insurance Claim application.

Entry.html

Customers enter the Policy Number to begin the Insurance Claim process in this form, as shown in Figure 17-3.

Figure 17-3 Entry.html



Welcome ! This is the Insurance Claim Process Management System.

To process your claim, you will need to provide information concerning your policy and detailed information about the claim itself. By providing this information online, we can quickly validate your information and process your claim as quickly as possible.

Please enter your Insurance Policy Number. This can be found on your policy itself, or on your latest policy account statement.

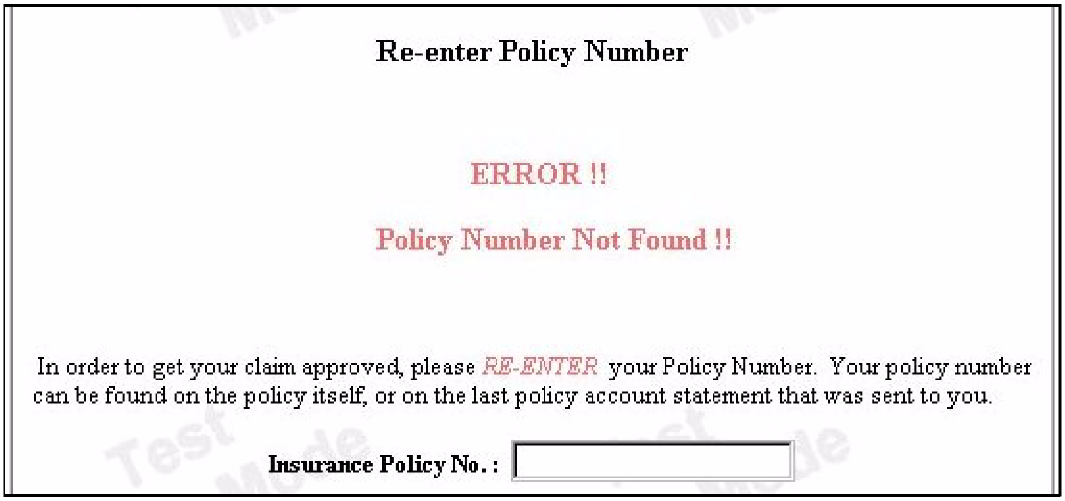
Insurance Policy No. : P12365

After completing the above form, select one of the following actions.

Resubmit.html

This form, shown in Figure 17-4, is displayed whenever the Policy Number, which the customer entered in Entry.htm, is not found in the flat file.

Figure 17-4 Resubmit.html



Re-enter Policy Number

ERROR !!

Policy Number Not Found !!

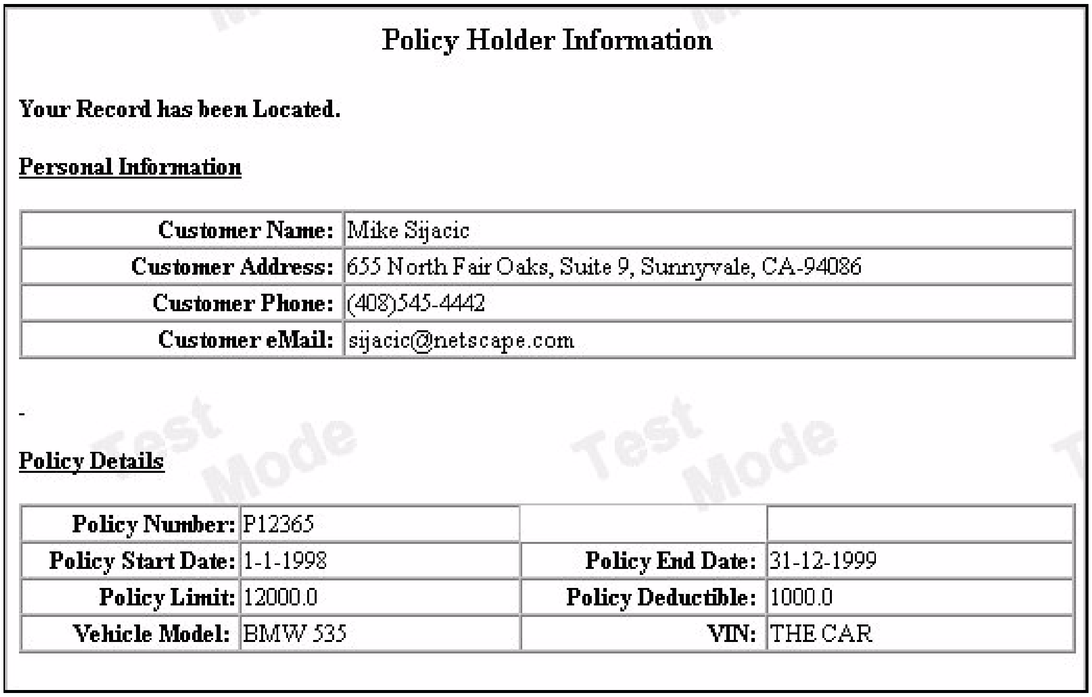
In order to get your claim approved, please *RE-ENTER* your Policy Number. Your policy number can be found on the policy itself, or on the last policy account statement that was sent to you.

Insurance Policy No. :

Details.html

This form, shown in Figure 17-5, displays the customer's personal information and the policy information that was found in the flat file.

Figure 17-5 Details.html



Policy Holder Information

Your Record has been Located.

Personal Information

Customer Name:	Mike Sijacic
Customer Address:	655 North Fair Oaks, Suite 9, Sunnyvale, CA-94086
Customer Phone:	(408)545-4442
Customer eMail:	sijacic@netscape.com

-

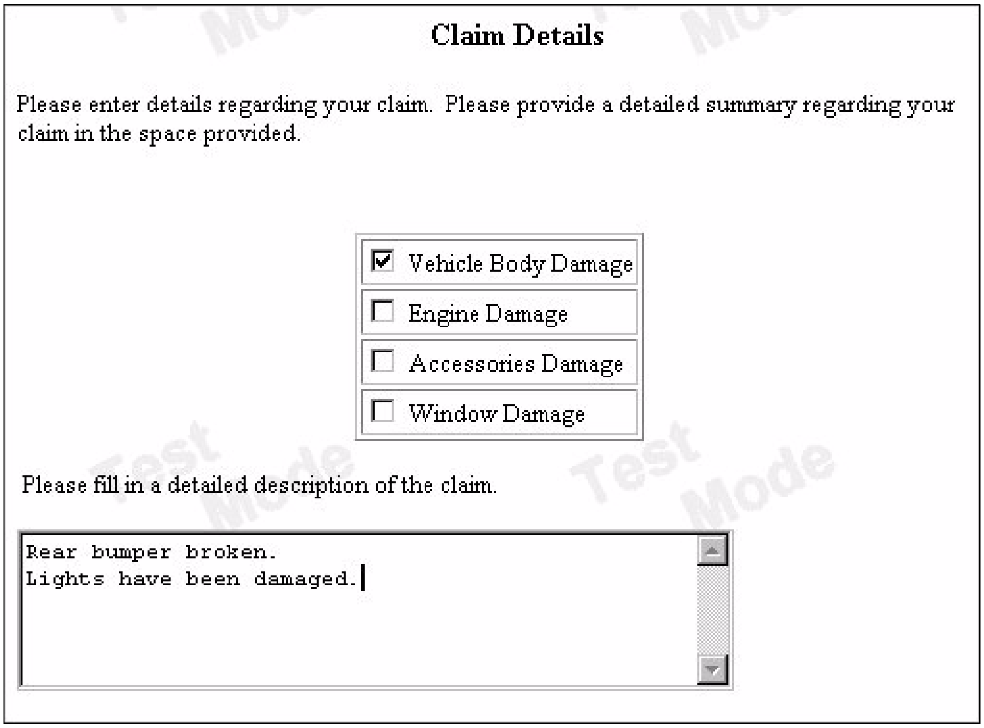
Policy Details

Policy Number:	P12365		
Policy Start Date:	1-1-1998	**Policy End Date:**	31-12-1999
Policy Limit:	12000.0	**Policy Deductible:**	1000.0
Vehicle Model:	BMW 535	**VIN:**	THE CAR

EnterClaim.html

Customers enter the claim in this form, shown in Figure 17-6. It has a predefined claim check boxes and a text area where the customer can enter a small description of the claim.

Figure 17-6 EnterClaim.html



Claim Details

Please enter details regarding your claim. Please provide a detailed summary regarding your claim in the space provided.

- [x] Vehicle Body Damage
- [ ] Engine Damage
- [ ] Accessories Damage
- [ ] Window Damage

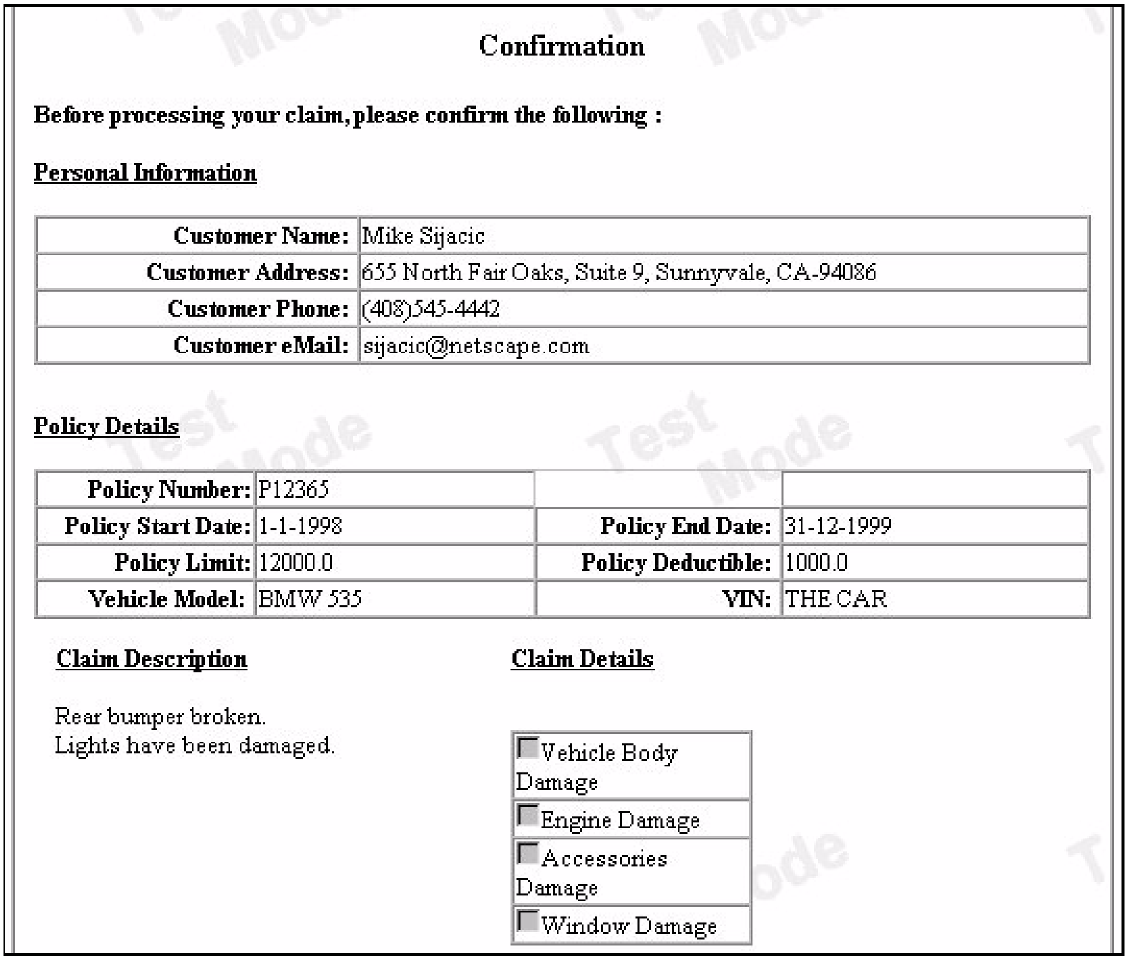
Please fill in a detailed description of the claim.

```
Rear bumper broken.
Lights have been damaged.
```

Confirmation.html

This form, shown in Figure 17-7, displays customer details from Details.html and claim details from EnterClaim.html.

Figure 17-7 Confirmation.html



Confirmation

Before processing your claim, please confirm the following :

Personal Information

Customer Name:	Mike Sijacic
Customer Address:	655 North Fair Oaks, Suite 9, Sunnyvale, CA-94086
Customer Phone:	(408)545-4442
Customer eMail:	sijacic@netscape.com

Policy Details

Policy Number:	P12365		
Policy Start Date:	1-1-1998	**Policy End Date:**	31-12-1999
Policy Limit:	12000.0	**Policy Deductible:**	1000.0
Vehicle Model:	BMW 535	**VIN:**	THE CAR

Claim Description

Rear bumper broken.
Lights have been damaged.

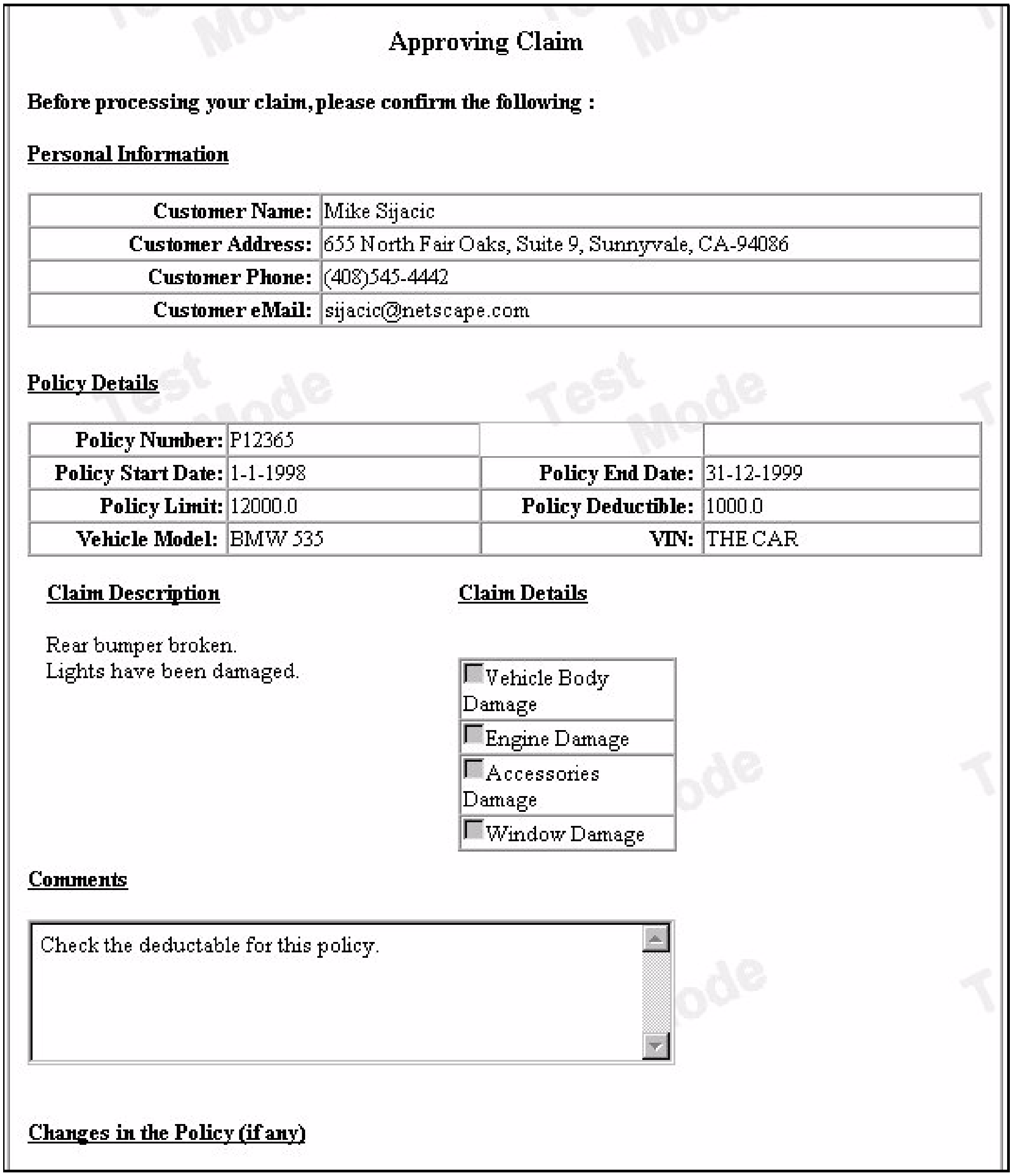
Claim Details

- ☐ Vehicle Body Damage
- ☐ Engine Damage
- ☐ Accessories Damage
- ☐ Window Damage

Approver.html

This form, shown in Figure 17-8, is same as Confirmation.html with an additional text area for the approver's comments.

Figure 17-8 Approver.html



Approving Claim

Before processing your claim, please confirm the following :

Personal Information

Customer Name:	Mike Sijacic
Customer Address:	655 North Fair Oaks, Suite 9, Sunnyvale, CA-94086
Customer Phone:	(408)545-4442
Customer eMail:	sijacic@netscape.com

Policy Details

Policy Number:	P12365		
Policy Start Date:	1-1-1998	**Policy End Date:**	31-12-1999
Policy Limit:	12000.0	**Policy Deductible:**	1000.0
Vehicle Model:	BMW 535	**VIN:**	THE CAR

Claim Description

Rear bumper broken.
Lights have been damaged.

Claim Details

- ☐ Vehicle Body Damage
- ☐ Engine Damage
- ☐ Accessories Damage
- ☐ Window Damage

Comments

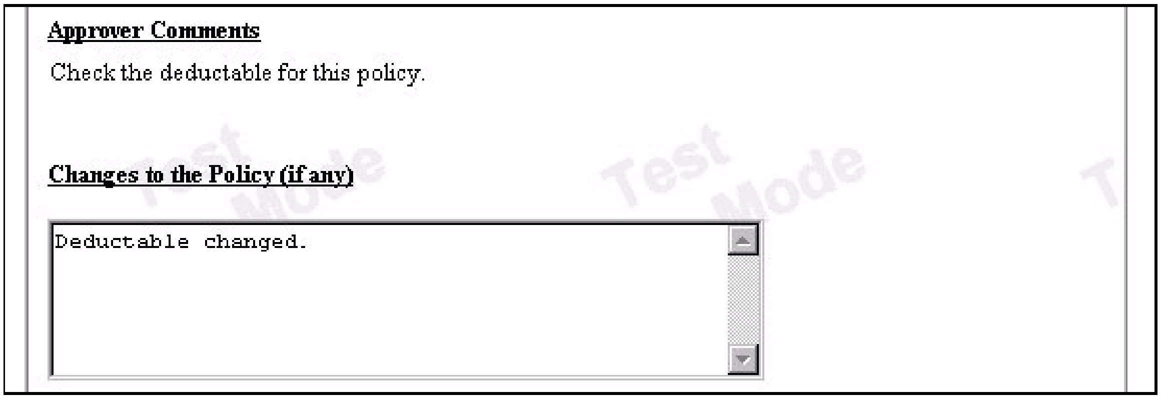
Check the deductable for this policy.

Changes in the Policy (if any)

Clarify.html

This form, shown in Figure 17-9, is similar to Approve.html but has an additional text area for the clarifier's comments.

Figure 17-9 Clarify.html



Approver Comments

Check the deductable for this policy.

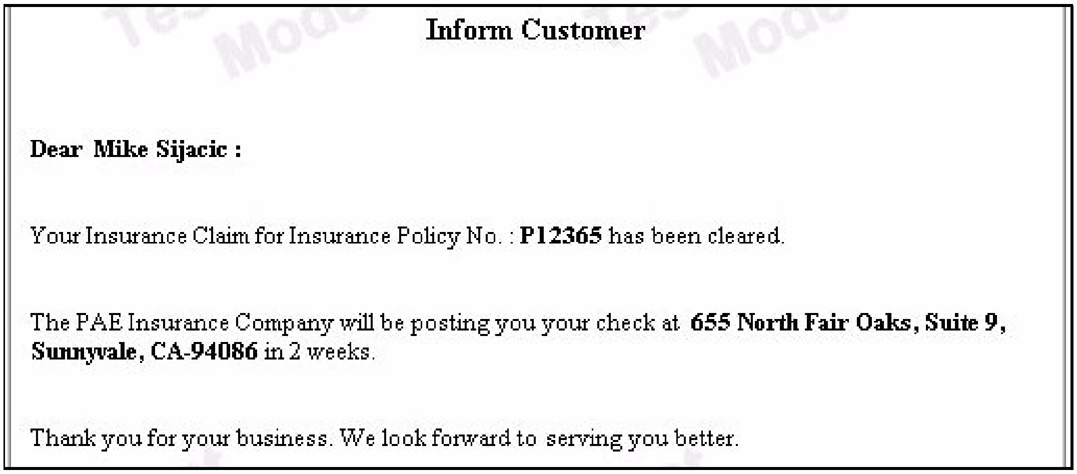
Changes to the Policy (if any)

```
Deductable changed.
```

InfCustomer.html

This form, shown in Figure 17-10, is displayed whenever a claim is approved. It shows the customer's name, customer's address, and the policy number.

Figure 17-10 InfCustomer.html



Inform Customer

Dear Mike Sijacic :

Your Insurance Claim for Insurance Policy No. : **P12365** has been cleared.

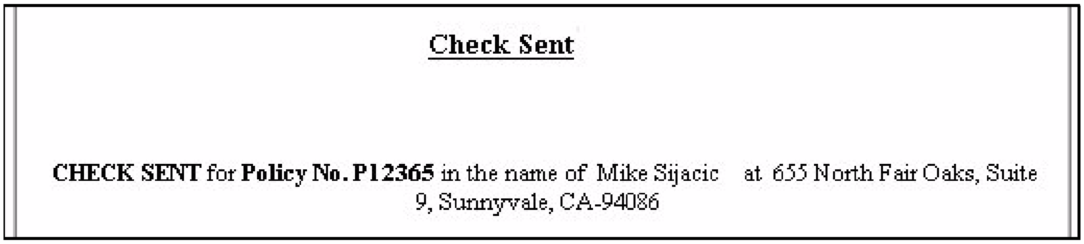
The PAE Insurance Company will be posting you your check at **655 North Fair Oaks, Suite 9, Sunnyvale, CA-94086** in 2 weeks.

Thank you for your business. We look forward to serving you better.

SendCheck.html

This form (see Figure 17-11) tells the customer that the claim check was sent.

Figure 17-11 SendCheck.html



Check Sent

CHECK SENT for **Policy No. P12365** in the name of Mike Sijacic at 655 North Fair Oaks, Suite 9, Sunnyvale, CA-94086

ClaimApproved.html

This exit form is displayed whenever a claim is approved and the process is successfully completed.

ClaimDenied.html

This exit form is displayed whenever a claim is denied and the process is successfully completed.

ExpirationExit.html

This exit form is displayed whenever a form has expired and it goes to the exit point.

Script Dictionary

The script dictionary contains the scripts used by the application. They are divided into categories based on the purpose they serve in the application.

Expiration Scripts

The expiration scripts run whenever a form has not been used or accessed for a period of time set by an expiration handler. When the period of time has elapsed without the form being accessed, the expiration handler script is executed.

The activities with expiration scripts in this application are set to expire in five minutes if the form is not accessed. In each activity's property window, the expiration setter is set to five minutes. Each activity has an expiration handler script that is called when the five minutes has passed. The script advances the process to the Expiration exit point.

The following sections show the expiration scripts at the activities where they occur:

Script at Policy Details Activity

```
function exppolicydetails ( )
{
  var pi = getProcessInstance ( );
  var wi = getWorkItem ( );
  // handle expiration
  // move process instance to Expiration Exit point.
  wi.setNodeName ( "Expiration Exit" ) ;
  return true;
}
```

Script at Enter Claim Details Activity

```
function expenterclaim ( )
{
  var pi = getProcessInstance ( );
  var wi = getWorkItem ( );
  // handle expiration
  // move process instance to Expiration Exit point.
  wi.setNodeName ( "Expiration Exit" ) ;
  return true;
}
```

Script at Confirmation

```
function expconfirmation ( )
  var pi = getProcessInstance ( );
  var wi = getWorkItem ( );
  // handle expiration
  // move process instance to Expiration Exit point.
  wi.setNodeName ( "Expiration Exit" ) ;
  return true;
}
```

Script at Resubmit Policy No.

```
function expresubmit ( )
{
  var pi = getProcessInstance ( );
  var wi = getWorkItem ( );
  // handle expiration
  // move process instance to Expiration Exit point.
  wi.setNodeName ( "Expiration Exit" ) ;
  return true;
}
```

Completion Scripts

Completion scripts are executed whenever an activity goes on to the next step. This is particularly helpful whenever you want to do a particular task such as displaying the form or setting some value in a data field as soon as the activity is completed.

In this application, after displaying at the policy details in Details.html, the application displays the EnterClaim.html form directly, without returning the user to their work item list. The completion script displays the form automatically to the user by using a function to redirect the URL.

The following sections show the completion scripts at the activities where they occur.

Completion Script at Policy Details Activity

```
function Enterclaim ( )
{
  var pi = getProcessInstance( );
  var process_id = pi.getInstanceId( );
  url =
"http://onguard/cgi-bin/gx.cgi/AppLogic+wfmySampleApp.npm?eventI
d=OnDisplayWorkItem&__instanceId="
  + process_id +
  "&__forkId=0&__nodeName=Enter%20Claim%20Details";
  setRedirectionURL(url);
  return true;
}
```

Completion Script at Enter Claim Details Activity

```
function Confirmation ( )
{
  var pi = getProcessInstance( );
  var process_id = pi.getInstanceId( );
  setRedirectionURL(
"http://onguard/cgi-bin/gx.cgi/AppLogic+wfmySampleApp.npm?eventI
d=OnDisplayWorkItem&__instanceId="
  + process_id +
  "&__forkId=0&__nodeName=Confirmation");
  return true; }
```

Required Files

The following sections show the additional required files for the Insurance Claim application.

The database.xml File

The following is an example of the database.xml file you need in order to run the application. The policy numbers listed in the file are the ones you need to enter in order to process an application.

```
<xml version="1.0" encoding="us-ascii">
<POLICYSET>

<POLICY>
  <CUSTOMER_NAME>Mike Sijacic</CUSTOMER_NAME>
  <CUSTOMER_ADDRESS>
      876 North Fair Oaks, Suite 9, Sunnyvale, CA-94086
  </CUSTOMER_ADDRESS>
  <CUSTOMER_PHONE_NUMBER>(408)123-4442</CUSTOMER_PHONE_NUMBER>
  <EMAIL> sijacic@netscape.com </EMAIL>
  <POLICY_NUMBER>P12365</POLICY_NUMBER>
  <START_DATE>1-1-1998</START_DATE>
  <END_DATE>31-12-1999</END_DATE>
  <POLICY_AMOUNT>12000</POLICY_AMOUNT>
  <POLICY_DEDUCTIBLE>1000</POLICY_DEDUCTIBLE>
  <VEHICLE_MODEL>BMW 535</VEHICLE_MODEL>
  <VEHICLE_NUMBER>THE CAR</VEHICLE_NUMBER>
</POLICY>

<POLICY>
  <CUSTOMER_NAME>MICHAL CHMIELEWSKI</CUSTOMER_NAME>
  <CUSTOMER_ADDRESS>
      3248 LAKEVIEW APTS,San Jose, CA-94086
  </CUSTOMER_ADDRESS>
  <CUSTOMER_PHONE_NUMBER>(408)123-4372</CUSTOMER_PHONE_NUMBER>
  <EMAIL> michal@netscape.com </EMAIL>
  <POLICY_NUMBER>P32345</POLICY_NUMBER>
  <START_DATE>1-1-1998</START_DATE>
  <END_DATE>31-12-1999</END_DATE>
  <POLICY_AMOUNT>12000</POLICY_AMOUNT>
  <POLICY_DEDUCTIBLE>1000</POLICY_DEDUCTIBLE>
  <VEHICLE_MODEL>BMW 747</VEHICLE_MODEL>
  <VEHICLE_NUMBER>seeCAR23</VEHICLE_NUMBER>
</POLICY>
```

```
<POLICY>
  <CUSTOMER_NAME>Souvik Das</CUSTOMER_NAME>
  <CUSTOMER_ADDRESS>
      844 Salt Lake, Moutain View, CA-94043
  </CUSTOMER_ADDRESS>
  <CUSTOMER_PHONE_NUMBER>(650)123-2343</CUSTOMER_PHONE_NUMBER>
  <EMAIL> souvik@netscape.com </EMAIL>
  <POLICY_NUMBER>P12346</POLICY_NUMBER>
  <START_DATE>1-1-1998</START_DATE>
  <END_DATE>31-12-1999</END_DATE>
  <POLICY_AMOUNT>12300</POLICY_AMOUNT>
  <POLICY_DEDUCTIBLE>400</POLICY_DEDUCTIBLE>
  <VEHICLE_MODEL>HONDA Accord V6 LE</VEHICLE_MODEL>
  <VEHICLE_NUMBER>PF12335</VEHICLE_NUMBER>
</POLICY>

<POLICY>
  <CUSTOMER_NAME>Albert TAM</CUSTOMER_NAME>
  <CUSTOMER_ADDRESS>884 Mary Blvd., Mountain View, CA-94043
  </CUSTOMER_ADDRESS>
  <CUSTOMER_PHONE_NUMBER>(650)123-4578</CUSTOMER_PHONE_NUMBER>
  <EMAIL> atam@netscape.com </EMAIL>
  <POLICY_NUMBER>P12347</POLICY_NUMBER>
  <START_DATE>1-1-1998</START_DATE>
  <END_DATE>31-12-1999</END_DATE>
  <POLICY_AMOUNT>12000</POLICY_AMOUNT>
  <POLICY_DEDUCTIBLE>1000</POLICY_DEDUCTIBLE>
  <VEHICLE_MODEL>BMW 535 v6</VEHICLE_MODEL>
  <VEHICLE_NUMBER>BPM GROUPIE3</VEHICLE_NUMBER>
</POLICY>

</POLICYSET>
```

The Policy.log File

The following is an example of the log file that is created when you run the application. It contains the ID numbers and claim information of process instances.

```
[Tue Aug 17 16:32:30 PDT 1999]
The Insurance Claim with Claim id : 1715745114  for Policy No. :
P12346 was initialized by : Souvik Das
[Tue Aug 17 16:33:27 PDT 1999]
Claim ID : 1715745114 With Claim : ’ Broken fuel tank ’ was
submitted
[Tue Aug 17 16:33:27 PDT 1999]
Claim ID : 1715745114 was APPROVED.
[Tue Aug 17 16:34:04 PDT 1999]
The Insurance Claim with Claim id : 618655790  for Policy No. :
P12365 was initialized by : Mike Sijacic
[Tue Aug 17 16:34:52 PDT 1999]
Claim ID : 618655790 With Claim : ’ All windows broken ’ was
denied
REASON : ’ null ’
```

The Banner Image

The application uses a banner image, shown in Figure 17-12, on all displayed forms.

Figure 17-12 The banner.gif file

The Background Image

The application uses a background image, shown in Figure 17-13, on all displayed forms.

Figure 17-13 The background image is called Gray_Textured1040.gif

Configuring the Insurance Claim Processing Application

Before running the Insurance Claim Processing application, make sure the `database.xml` file resides in the correct location in your environment. This file contains sample policy data. The application retrieves this data for a policy that the user enters.

On Solaris, `database.xml` must reside in the home directory of the user who is running the application server. On Windows NT, you must place `database.xml` in the root of the drive of where Windows NT is installed. For example, if Windows NT is installed on the `E:` drive, then `database.xml` file must reside at `E:\database.xml`.

Note that the policy.log file will be created in the same location as `database.xml`.

Custom Activity Code

This section summarizes the three custom activities in the Claim Processing sample application:

- "The LogPerformer Activity"
- "The LogdenialPerformer Activity"
- "The LookupPerformer Activity"

For information on writing and packaging a custom activity, see the *Process Manager Programmer's Guide*.

The LogPerformer Activity

The LogPerformer custom activity writes specific information to a log file. This custom activity is represented by the `LogPerformer.jar` archive, which contains the following files:

`/LogPerformer.xml`

`/com/model/sampleapp/logperformer/Logger.class`

`/com/model/sampleapp/logperformer/LogPerformer.class`

The LogdenialPerformer Activity

The LogdenialPerformer custom activity writes a claim process denial to a log file. This custom activity is represented by the `LogdenialPerformer.jar` archive, which contains the following files:

```
/LogdenialPerformer.xml
/com/model/sampleapp/logdenialperformer/Logger.class
/com/model/sampleapp/logdenialperformer/LogdenialPerformer.class
```

The LookupPerformer Activity

The LookupPerformer custom activity takes the policy number as input, searches a database of policies, and returns the policy details back to the process. This custom activity is represented by the `LookupPerformer.jar` archive, which contains the following files:

```
/LookupPerformer.xml
/com/model/sampleapp/lookupperformer/Logger.class
/com/model/sampleapp/lookupperformer/LookupPerformer.class
/com/model/sampleapp/lookupperformer/MyParser.class
/com/model/sampleapp/lookupperformer/Policy.class
/com/model/sampleapp/lookupperformer/PolicySet.class
```

Code Walkthrough for LookupPerformer.java

This section describes the steps for creating the `LookupPerformer.java` file. This file contains three main parts:

- "Definitions and Packages"
- "The init, perform, and destroy Methods"
- "The GetPolicy Method"

Definitions and Packages

The first part of `LookupPerformer.java` is written as follows:

1. Define a package for your class:

```
   package com.model.sampleapp.lookupperformer;
```

2. Import the required Java packages:

```
   import java.lang.*;
   import java.util.*;
   import com.model.sampleapp.lookupperformer.*;
   import com.netscape.pm.model.*;
   import java.io.*;
```

3. A custom activity must implement the `ISimpleWorkPerformer` interface:

```
   public class LookupPerformer implements
   com.netscape.pm.model.ISimpleWorkPerformer
 {
```

4. Next, define the variables to use within the custom activity:

```
  private String CustomerName ;
  private String CustomerAddress ;
  private String CustomerPhoneNo ;
  private String CustomerEmail ;
  private double PolicyAmt;
  private String PolicyNo ;
  private String StartDate ;
  private String EndDate ;
  private double Deductible ;
  private String VehicleModel ;
  private String VIN  ;

  private String mFileName ;
```

The init, perform, and destroy Methods

1. The `init` method is used to set up anything you'll need globally within the custom activity. In this example, the method initializes the name and location of the file to use to search for policies.

```
public void init( Hashtable env ) throws Exception {
            mFileName = "database.xml";
        }
```

2. The `perform` method does the main work of the custom activity.

```
public void perform( Hashtable input, Hashtable output )
  throws Exception {
```

This method takes data from an input hashtable (which is passed in from the process), does the necessary work, and then puts the data back into an output hashtable (which is passed back to the process).

3. The data hashtable from the process is in the `input` variable. Similarly, the hashtable in which to pass data back to the process is in the `output` variable.

```
  // Read the Policy_No attribute from the input hashtable
  Object value = input.get ( "Policy_No" ) ;
```

4. Next, you check to see whether the value from the hashtable is in the proper format, a string.

```
  if ( value instanceof com.netscape.javascript.NativeJavaObject
)
  {
  input.put ("Policy_No",
    ((com.netscape.javascript.NativeJavaObject)
      value).unwrap().toString()) ;
  }
  String mPolicyNo = (String) input.get( "Policy_No" );
```

5. A call to `GetPolicy` will parse the `database.xml` file and return the details of the policy.

```
  Policy ourPolicy = GetPolicy (mPolicyNo);
```

6. Check whether the policy number received by the custom activity is blank. If so, return an empty policy number in the output hashtable. You'll check whether the policy number returned within the process is empty and show an error screen there.

```
  if ( ourPolicy == null ){
      // throw new Exception("Policy No not found");
      PolicyNo = " ";
      output.put ( "PolicyNo", PolicyNo ) ;
      return;
  }
```

7. Next, generate a random number to assign as the claim number.

```
String RanNum = null ;
Random rgen = new Random ();
RanNum = Integer.toString (rgen.nextInt());

if ( RanNum.startsWith ("-") ) {
    RanNum = RanNum.substring ( 1 ) ;
}
```

8. Now construct a message line for logging. The message contains the claim number and some of the policy details.

```
String PNo = (String) input.get( "Policy_No" );
StringBuffer temp1 = new StringBuffer
    ( "The Insurance Claim with Claim id : " );
temp1.append ( RanNum )
temp1.append ( "  for Policy No. : " + PNo);
CustomerName = ourPolicy.getCustomerName ( ) ;
temp1.append ( " was initialized by : " + CustomerName );
```

9. Open a Logger object that writes the information to a log file.

```
Logger logger = new Logger ();
logger.open();
logger.log ( temp1.toString() ) ;
logger.close();
```

10. Define variables based on the `ourPolicy` object, which contains the details of the customer and policy.

```
  CustomerName = ourPolicy.getCustomerName ( ) ;
  CustomerAddress = ourPolicy.getCustomerAddress ( ) ;
  CustomerPhoneNo = ourPolicy.getCustomerPhoneNo ( ) ;
  CustomerEmail = ourPolicy.getCustomerEmail ( ) ;
  PolicyNo = ourPolicy.getPolicyNo( ) ;
  PolicyAmt = ourPolicy.getPolicyAmt( ) ;
  StartDate = ourPolicy.getStartDate( ) ;
  EndDate = ourPolicy.getEndDate( ) ;
  Deductible = ourPolicy.getDeductible( ) ;
  VehicleModel = ourPolicy.getVehicleModel( );
  VIN = ourPolicy.getVIN( );
```

11. Next, populate the output hashtable with the information from the policy.

```
  // Set the output hashtable
  output.put ( "CustomerName", CustomerName ) ;
  output.put ( "CustomerAddress", CustomerAddress ) ;
  output.put ( "CustomerPhoneNo", CustomerPhoneNo ) ;
  output.put ( "CustomerEmail", CustomerEmail ) ;
  output.put ( "PolicyNo", PolicyNo ) ;
  output.put ( "PolicyAmt", Double.toString(PolicyAmt) ) ;
  output.put ( "StartDate", StartDate ) ;
  output.put ( "EndDate", EndDate ) ;
  output.put ( "Deductible", Double.toString(Deductible) ) ;
  output.put ( "VehicleModel", VehicleModel ) ;
  output.put ( "VIN", VIN ) ;
  output.put ( "ClaimId", RanNum );
// End of perform method
}
```

12. The `destroy` method is invoked when the application is unloaded from the application server. It is the opportunity to clean up resources that are used by the server. In this case, there is nothing to do.

```
  public void destroy( ){
  }
```

The GetPolicy Method

The `GetPolicy` method returns a policy object that contains the information on a given policy. The method itself opens up the `database.xml` file and parses through it to find the appropriate policy. Here is the code for this method:

```
private Policy GetPolicy (String PNO)
{
  try
  {
      String fileName = "database.xml";
      int Pfound = 0;


      int  MAX_LENGTH = 50000 ;
      char [] xml = new char [ MAX_LENGTH ] ;


      java.io.File f = new java.io.File ( fileName ) ;
      FileReader fr = new FileReader ( f  ) ;
      BufferedReader in = new BufferedReader( fr );
      int count = 0 ;
      while ((count = in.read (xml, count, MAX_LENGTH )) != -1 )
      {}
      String xmlStr = new String ( xml ) ;
      MyParser parser = new MyParser ( xmlStr ) ;
      PolicySet tempPolicySet = parser.parse ( ) ;
      int j;

      Policy policy = null ;
      for ( j=0 ; j < tempPolicySet.GetSize(); j++ )
      {
          policy = (Policy) tempPolicySet.elementAt(j) ;
          if ( policy.getPolicyNo().equals (PNO ) )
          {
              Pfound = 1 ;
              System.out.println("PNO in GET POLICY IS  :  " PNO
);
              System.out.println( "Policy Not Found!!");
              return policy;
          }
      }
      return null;      // If Policy not found ...


      } catch ( Exception ignore )
      {
        return null;
      }
  }

// end class
}
```

Appendix A

JavaScript API Reference

This appendix is an API reference of the objects and methods you can use when writing JavaScript scripts for Process Manager.

The Process Manager Object Model includes the following objects:

- "ProcessInstance" on page 386 contains methods for accessing information about the current process.
- "WorkItem" on page 392 contains methods for accessing information about the current work item.
- "ContentStore" on page 398 contains methods for accessing the content store.
- "CorporateDirectory" on page 412 contains methods for accessing data about users in the Directory Server.
- "User" on page 419 exposes the user attributes that make up an LDAP user entry as publicly accessible properties.

The following global functions are also available for use in scripts:

- "Logging and Error Handling Global Functions" on page 420
- "Assignment, Completion, and Email Scripts" on page 424
- "Miscellaneous Global Functions" on page 433

All functions and methods are listed here:

- "Alphabetical Summary of JavaScript Methods and Functions" on page 452

ProcessInstance

A script can use the global function `getProcessInstance()` to get the processInstance for process in which the script is running.

Example:

```
var pi = getProcessInstance ();
```

The following methods can be called on the processInstance object:

- getCreationDate
- getCreatorUser
- getData
- getEntityKey
- getEntryNodeName
- getExitNodeName
- getInstanceId
- getPriority
- getRoleUser
- getTitle
- setData
- setEntityKey
- setRoleById

getCreationDate

Returns a JavaScript date that represents the date when the process instance was first created.

Example:

```
var pi = getProcessInstance();

var creationDate = pi.getCreationDate();
```

getCreatorUser

Returns an object that contains the attributes of the creator user in the corporate directory.

Example:

```
var pi = getProcessInstance ();

var creator = pi.getCreatorUser ();
```

getData

Returns the value of the data field or undefined if the field is not defined. The returned value is in JavaScript format.

Parameters

• *fieldname*	String of the field name whose value is being retrieved

Example

```
var pi = getProcessInstance ();

var cust_name = pi.getData( "customer_name" );
```

getEntityKey

Returns the entity key associated with the custom field whose name is passed as an argument. This key is stored in the process instance and is set using the `setEntityKey` method. The entity key is the primary key for the field, uniquely identifying it from other instances of the same custom field.

Parameters

• *fieldName*	String of the name of the custom field for which you want to fetch the entity key.

Example:

```
var pi = getProcessInstance();

pi.getEntityKey("mySignature");
```

getEntryNodeName

Returns a JavaScript string containing the name of the entry point node where the current process instance was initiated. All process instances must start from an entry node, ensuring that this method never returns `null`.

Example:

```
var pi = getProcessInstance();

var entry = pi.getEntryNodeName() );
```

getExitNodeName

Returns a JavaScript string containing the name of the exit point node where the process instance has completed. If the process instance has not yet completed, returns `null`.

Example:

```
var pi = getProcessInstance();

var exitNodeName = pi.getExitNodeName();
```

getInstanceId

Returns the ID of the process instance, such as `"2345".` The id is generated by the engine to be unique across all process instances of all applications defined in the same cluster.

Example:

```
var pi = getProcessInstance();

var pid = pi.getInstanceId();
```

getPriority

Returns a JavaScript number representing the priority value of the current process instance. The priority value ranges from 1 (highest) to 5 (lowest). The priority value is derived from the value of a data element. If the data element's type is INT/FLOAT, the priority value is fetched from the data element's value. Otherwise, the default value 3 is returned.

Example:

```
var pi = getProcessInstance();

var priority = pi.getPriority();
```

getRoleUser

Returns a JavaScript User object that contains the attributes of the user that has been assigned to the role `roleName`. If a user is not associated with the role roleName yet, null is returned.

Parameters

• *roleName*	String of the name of the role whose full DN is returned.

Example:

```
var pi = getProcessInstance ();

var u = pi.getRoleUser( "role1" );
```

getTitle

Returns the current title string associated with the process instance. Calling this method is similar to calling `getData()` on the title data field and casting the value to a string. If the title data field has a type other than TEXT/LONG TEXT, returns a textual representation of the data element's value.

Example:

```
var pi = getProcessInstance();

var title = pi.getTitle();
```

setData

Associates the object *fieldValue* with the data element *fieldName*. The object passed by *fieldValue* can be any valid JavaScript object (such as String, Date, Array). The object can be a Java object if the data element is a custom field.

Parameters

• *fieldName*	String of the name of the data element whose value is modified.
• *fieldValue*	Object which is the value associated with the data element `fieldName`.

Since each data element is configured with a particular data type which is set at design time in Process Builder, be aware of the following data conversion guidelines:

Data element type	fieldValue type	fieldValue converted to...
TEXT/LONGTEXT	any type	String
INT	String	int
INT	Integer	int
FLOAT	String	float
FLOAT	Integer	float
DATE/DATETIME	String	Date
DATE/DATETIME	Date	Date

Example:

```
var pi = getProcessInstance ();
pi.setData( "customer_name", "Joe" );
//Set value for custom field
//
var custAcct = new Packages.com.acme.Account();
pi.setData( "customer_acct", custAcct );
```

setEntityKey

Sets the entity key of a custom field. The entity key is the primary key for the field, uniquely identifying it from other instances of the same custom field. The entity key is always available from the process instance, and is stored with the rest of the application data.

Parameters

• *fieldName*	String of the name of the custom field you want to modify.
• *value*	Object which is the value of the custom field's entity key.

Example:

```
var pi = getProcessInstance() ;

pi.setData ("mySignature" , pi.getData("signature") ) ;

pi.setEntityKey("mySignature" , "bob") ;
```

setRoleById

Changes the user associated with the role *roleName* by specifying the new user's ID via *userId*. Returns `true` if *userId* corresponds to an actual user in the corporate directory and the user associated with *roleName* is changed. Returns `false` if *userId* does not correspond to an actual user entry and *roleName* is left unchanged.

Parameters

• *roleName*	String of the name of the role to alter.
• *userDN*	String of the Id of the user to associate with roleName.

Example:

```
var pi = getProcessInstance ();

pi.setRoleById( "myRoleName", "joe" );
```

WorkItem

A script can use the global function `getWorkItem()` to get the work item in which the script is running.

Example:

```
var wi = getWorkItem();
```

The following methods can be called on the workItem object:

- addUserAssignee
- addGroupAssignee
- assignees
- extend
- getCreationDate
- getExpirationDate
- getNodeName
- hasExpired
- isStateActive
- isStateRunning
- isStateSuspended
- isUserAssignee
- moveTo
- removeAssignees
- resume
- setExpirationDate
- suspend

addUserAssignee

`addUserAssignee(userID)`

Adds the given user as an assignee for the work item.

Parameters

• `userID`	User ID of the user to be added as an assignee.

The following script adds the user whose uid is `jocelynb` as an assignee to the work item.

Example:

```
var wi = getWorkItem();
wi.addUserAssignee("jbecker");
```

addGroupAssignee

`addGroupAssignees(groupName)`

Adds the members of the given group as assignees for the work item.

Parameters

• `groupName`	Name of the role.

assignees

This method returns a JavaScript array of the user ids of the assignees of a work item. Since automated activities are performed by Process Engine and do not require an assignee, use this method only from a user activity.

Example:

```
var wi = getWorkItem();
var assignees = wi.assignees();

for( var i = 0; i < assignees.length; i++)
  doSomethingForEachAssignee(assignees[ i ] ;
```

extend

Extends the duration of the work item by pushing out its expiration date. If the work item was expired, this function removes the expired flag.

Parameters

• *newDate*	Date object for the new expiration date for the work item.

Example:

```
var wi = getWorkItem();

var now = new Date();

// Set extension date to 1 hour in the future

var newDate = new Date( now.getTime() + (1 * 60 * 60 * 1000) );

wi.extend( newDate );
```

getCreationDate

Returns a JavaScript date that represents the creation date of the work item.

Example:

```
var wi = getWorkItem();

var wiCreationDate = wi.getCreationDate();
```

getExpirationDate

Returns a JavaScript `Date` object that represents the expiration date of the work item.

Example:

```
var wi = getWorkItem();

var expDate = wi.getExpirationDate();
```

getNodeName

Returns the name of the workitem.

hasExpired

Returns `true` if the work item has expired or `false` if it has not expired. If the work item has already expired, but has been extended via the `extend()` method, `hasExpired()` returns false if called again.

Example:

```
var wi = getWorkItem();

// if the work item has not expired, then expire it in 1 minute

if( wi.hasExpired() == false )

   expireIn( 1, "minutes" );
```

isStateActive

Returns `true` if the work item is active or `false` if the work item is inactive. A work item is considered active if there is only one assignee currently assigned to it. If a work item has been assigned to a group, the work item is not active, but running. Once one member of the group elects to work on the item, it is considered active.

isStateRunning

Returns `true` if the work item is running or `false` if it is not running. A work item is running if more than one user is assigned to it. Once a user starts the work item, the work item changes state from running to active.

isStateSuspended

Returns `true` if the work item is suspended or `false if` it is not suspended. You can suspend a work item through Process Administrator or by calling `suspend()`.

Example:

```
var wi = getWorkItem();

if( wi.isStateSuspended() )

   wi.resume(); //resume the work item immediately
```

isUserAssignee

Tests whether a user is an assignee of the work item or not.

Parameters

• *userID*	String of the user ID.

Example:

```
// If chowdary is not an assignee, add him as one
var wi = getWorkItem();
if (wi.isUserAssignee("chowdary") == false)
   wi.addUserAssignee("chowdary")
```

moveTo

Move the current work item to the activity specified in *activityName*. Only activities associated with a virtual transition can use this method. `moveTo()` can only be called from expiration handler scripts and completion scripts.

Note that `moveTo()` doesn't actually involve "moving." Rather, the current work item is destroyed and a new work item is created at the new activity.

Parameters

• *activityName*	String of the name of the activity to which to move the work item.

Example:

```
var wi = getWorkItem(); //inside the expiration handler script
var activityName = "expired item";
wi.moveTo( activityName );
```

removeAssignees

`removeAssignees()`

Removes all assignees from the workitem.

Example:

The following script removes all existing assignees from a work item and adds the user whose ID is `jocelynb` as the sole assignee.

```
var wi = getWorkItem();
wi.removeAssignees();
wi.addUserAssignee("jocelynb");
```

resume

This function reactivates a suspended work item. If the work item is assigned to more than one user, the work item's state is set to running. If there is only one assignee, the work item's state is set to active.

Example:

```
var wi = getWorkItem();

if( wi.isStateSuspended() )

   wi.resume(); //resume the work item immediately
```

setExpirationDate

Sets the expiration date of the work item. If the work item has expired, use the `extend()` function instead of `setExpirationDate()` to extend the expiration date, since `extend()` removes the expired flag and `setExpirationDate()` does not.

Parameters

- `expDate`   Date object for the expiration date for the work item.

Example:

```
var wi = getWorkItem();
var now = new Date();

//Set expiration date to 1 hour in the future

var expDate = new Date( now.getTime() + (1 * 60 * 60 * 1000)
);
wi.setExpirationDate( expDate );
```

suspend

This function makes the work item inactive. Users cannot work with the work item until it has been resumed through Process Administrator or via a call to `resume()`.

Example:

```
var wi = getWorkItem();
wi.suspend();
```

ContentStore

Scripts can use the global function `getContentStore()` to get a `contentStore` object connected to the content store of the application. There are three different ways to call the function:

- `getContentStore ();`

  Without parameters, this function returns the content store defined by the application. The returned object is configured with the URL, username, and password that are specified in Process Builder.

- `getContentStore (httpURL, username, password);`

  When called with the above three parameters, `getContentStore()` creates a new content store with the specified parameters passed in. The returned object is a content store rooted at the specified httpURL. The content store uses the username and password to authenticate against.

- `getContentStore (httpURL, username, password, timeout);`

This form is identical to the previous form except for an additional timeout, specified in seconds, on HTTP connections.

The following methods can be called on the `contentStore` object:

- copy
- download
- exists
- getBaseURL
- getBaseURL
- getBaseURL
- getRootURL
- getContent
- getException
- getSize
- getStatus
- getVersion
- initialize
- isException
- list
- mkdir
- move
- remove
- rmdir
- store
- upload

copy

`copy(srcURL, dstURL)`

Copies the content at the *srcURL* to the *dstURL*. Both URLs have to be on the same content store. If *dstURL* already exists, it is overwritten. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

- `srcURL` A String of an HTTP URL or URI of the content in the content store object.
- `dstURL` A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();
var result = cs.copy( cs.getBaseURL ( "shopping/docs" ) +
  "myFile.html",
var httpcode = cs.getStatus ( result );
if (httpcode == 200)
{
  // it went OK, so continue
}
```

download

`download(url, file)`

Downloads the content at the url into the specified file. If a username and password is specified for this content store, they are used in the request to retrieve the content. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

- *url* A String of an HTTP URL or URI of the content in the content store object.
- *file* File where the content is stored; if exists, it will be overwritten.

Example:

```
var cs = getContentStore();
var result = cs.download ( cs.getBaseURL() + "myFile.html" ,
  "C:/myFile.html" );
var httpcode = cs.getStatus( result );
if (httpcode == 200)
{
  // it went OK so continue
}
```

exists

`exists(url)`

Checks if the URL exists on the content store. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

- *url* A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();
var result = cs.exists( cs.getBaseURL ( "shopping/docs" ) +
"myFile.html" );
var httpcode = cs.getStatus ( result );
if (httpcode == 200)
{
  // file exists,so continue
}
```

getBaseURL

`getBaseURL()`

Returns the root URL of this content store.

See `getRootURL` for examples.

getBaseURL

`getBaseURL(path)`

Returns a url which appends the given path to the content store root url.

Parameters

• *path*	A String of the relative path segment.

Example:

```
var cs = getContentStore();

var shoppingDocBaseURL = cs.getBaseURL("shopping/docs");
```

getBaseURL

`getBaseURL(path, instanceid)`

Returns a URL which appends the path and the `instanceid` to the content store root URL.

Parameters

• *path*	A String of a relative path segment.
• *instanceid*	An integer instance id of the process.

Example:

```
var pi = getProcessInstance();

var cs = getContentStore();

var myURL= cs.getBaseURL("shopping/docs", pi.getInstanceId());
```

getContent

`getContent (url)`

Returns a String of the content at the specified url, where `url` describes either an HTTP URL or URI of the content in the content store object. A URI is a path segment only, specified as an absolute or relative pathname.

Parameters

• *url*	A String of an HTTP URL or URI of the content in the content store object.

On an HTTP content store, getContent( ) is implemented as a GET request against the server. If a username and password have been specified for this content store, they will be used in the request to retrieve the content.

If the content cannot be retrieved or doesn’t exist, getContent( ) returns null. Call the exists( ) method to check whether the URL exists in the content store. See also store( ) and upload( ).

Example:

```
var cs = getContentStore();
// Relative URI
var content = cs.getContent ( "myFile.html" );

// Absolute URI
var content = cs.getContent ( "/CS/CheckOut/basket.xml" );

// If URL, then the host and port portion of the URL must be
// that of the content store URL. This is typically returned
// by the getBaseURL() method.
var content = cs.getContent ( cs.getBaseURL() +
  "CheckOut/basket.xml" )
```

getException

`getException(result_string)`

Gets the exception in string format that has occurred in the operation `move()`, `copy()`, `upload()`, `download()`, `store()`, `exists()`, `initialize()`, `remove()`, `mkdir()`, `rmdir()`.

Parameters

- *result_string* A String of the results from one of the methods listed above.

Example:

```
var cs = getContentStore();
// Remove a file from the content store
var fileName = cs.getBaseURL ("shopping/docs") + "myFile.html" )
var result = cs.remove(fileName);

// If something went wrong, put an error in the error log
if ( cs.isException (result))
{
  var e = new Object();
  e.exceptionString = cs.getException(result);
  logErrorMsg ("FILE_REMOVAL_ERROR", e);
}
```

getRootURL

`getRootURL ()`

Returns a string of the root URL of the content store, including the ending `/`. This is a string version of the URL for the content store as specified in Process Builder.

Example:

```
var cs = getContentStore();

var rootURL = cs.getRootURL();
```

getSize

`getSize(url)`

Returns the size in bytes of the content of the specified URL at the content store. On any error, -1 is returned.

Note that in certain rare circumstances, the length of the content of such a URL cannot be established.

Parameters

- *url* A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();

var size = cs.getSize( cs.getBaseURL ( "shopping/docs" ) +
"myFile.html" );
```

getStatus

`getStatus(result)`

Returns the status of the last successful Content Store operation by one of the methods: `move()`, `copy()`,`upload()`,`download()`,`store()`,`exists()`, `initialize()`,`remove()`,`mkdir()`,`rmdir()`.

Parameters

• *result*	A String of the results of one of the methods listed above.

Example:

```
var cs = getContentStore();
var result = cs.remove( cs.getBaseURL ( "shopping/docs" ) +
"myFile.html" );
if ( cs.getStatus(result) != 200)
{
  // there was an error, so do something,
  // perhaps put an error in the error log
}
```

getVersion

`getVersion()`

Returns the version string of the ContentStore implementation. Currently, this value is 0.1999100221-001.

Example:

```
var cs = getContentStore();

var version = cs.getVersion();
```

initialize

`initialize(url)`

Initializes the URL with basic content so the URL exists on the Content Store. This method returns either the status code from the content store for the operation or an exception. Note that no exception is actually thrown. To see if an exception is returned, call `isException()` on the return value from this method and call `getStatus()` to get the numeric HTTP protocol status of this content store request.

Parameters

• *url*	A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();

var myURL = cs.getBaseURL("shopping/docs") + "myFile.html";
var result = cs.initialize (myURL);
var httpcode = cs.getStatus ( result );

// 201 or 204 indicates an error
if (httpcode == 201 || httpcode == 204)
{
  var error1 = new Object();
  error1.operation = "initialization";
  error1.URL = myURL;
  logErrorMsg ("URL_INITIALIZATION_ERROR", error1);
}
```

isException

`isException(exception)`

Checks if the result string from `move()`, `copy()`, `upload()`, `download()`, `store()`, `exists()`, `initialize()`, `remove()`, `mkdir()`, `rmdir()` is an exception.

Parameters

- *exception* A String of the results of one of the methods of Content Store that can return an exception.

Example:

```
var cs = getContentStore();

var myURL = cs.getBaseURL ( "shopping/docs" ) + "myFile.html";
var result = cs.remove( myURL);

// If an exception occurred, put an error in the error log
if ( cs.isException ( result ))
{
  var error1 = new Object();
  error1.operation = "removal";
  error1.URL = myURL;
  logErrorMsg ("CONTENT_STORE_REMOVAL_ERROR", error1);
}
```

list

`list(url)`

Gets the container content (folder) listing of this content store container.

Parameters

- *url* A String of an HTTP URL or URI of the content in the content store object.

mkdir

`mkdir (URLString)`

Creates the appropriate folder structure on the web server being used as the content store. The *URLString* must be a full path name to the new directory. If the specified directory already exists, nothing happens. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

- *URLString* A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();

// Make a new directory in the content store
var dirName = "shopping/docs";
var result = cs.mkdir( cs.getBaseURL ( dirName) );

// If an exception occurred, put an entry in the error log
if (cs.isException ( result ))
{
  var error1 = new Object();
    error1.directoryName = dirName;
  logErrorMsg ("CONTENT_STORE_MAKE_DIRECTORY_ERROR", error1);
}
```

move

`move (URLString1, URLString2)`

moves the file from the *URLString1* to *URLString2*, overwriting any file that already exists at the destination file name. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

If the file to be moved *URLString1*, does not exist, this function gives an error.

Both file names must be full pathnames. The following code moves the file `tempdoc.html` from the `temp` directory to `doc1.html` in the `docs` directory on the web server `pm.company.com`.

Example:

```
var myStore = getContentStore ();

var pi = getProcessInstance ();

var pid = pi.getInstanceId();

var result = myStore.move("http://pm.company.com/temp/tempdoc.html",

       "http://pm.company.com/docs/doc1.html");
```

Note that the `move()` function does not create directories -- use `mkdir()` to create a directory before moving a file to it.

remove

`remove(url)`

Deletes the content at the specified URL. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

- *url* A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();
var myURL = cs.getBaseURL ("shopping/docs") + "myFile.html"
var result = cs.remove(myURL);

// If an error occurred, put an entry in the error log
if (cs.isException (result) )
{
  var error1 = new Object();
  error1.operation = "removal";
  error1.URL = myURL;
  logErrorMsg ("CONTENT_STORE_REMOVAL_ERROR", error1);
}
```

rmdir

`rmdir(url)`

Removes the container (folder) at the specified URL. The folder must be empty to be removed or an error is returned. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

- *url* A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();

var result = cs.rmdir( cs.getBaseURL ( "shopping/docs" ) );
```

store

`store (content, url)`

Stores the content specified by *content* in the given url. If a username and password has been specified for this content store, they are used in the request to store the content. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

`content`	A String of the content to store in the content store as this URL.
`url`	A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();
var content = "<body><b>Hello</b> world ... </body>";
// Relative URI
var result = cs.store (content, "myFile.html" );
// Absolute URI
var result = cs.store (content, "/CS/myFile.html" );
// Absolute URL (for historical reasons)
var result = cs.store (content, cs.getBaseURL() + "myFile.html");
```

upload

`upload(file, url)`

Uploads the file content to the URL specified. If a username and password has been specified for this content store, they are used in the request to upload the file. Returns a result string that can be passed to content store methods such as `getStatus()`, `getException()` and `isException()` to test for the result of the operation.

Parameters

`file`	File to upload.
`url`	A String of an HTTP URL or URI of the content in the content store object.

Example:

```
var cs = getContentStore();
```

```
var result = cs.upload ( "C:/myFile.html", cs.getBaseURL() +
"myFile.html" );
```

CorporateDirectory

Scripts can use the global function `getCorporateDirectory()` to get a `corporateDirectory` object that is connected to the corporate directory in the Directory Server for the cluster.

Example:

```
var corp = getCorporateDirectory ();
```

Scripts can then use the `corporateDirectory` object to access and modify information about users in the corporate directory.

The following methods can be called on the `corporateDirectory` object:

- addUser
- deleteUserByCN
- deleteUserByDN
- deleteUserById
- getUserByCN
- getUserByDN
- getUserById
- modifyUserByCN
- modifyUserByDN
- modifyUserById

addUser

`addUser (userDN, attributes, objectClasses)`

Adds a user to the directory. The user's Distinguished Name (DN) is specified by *userDN*. The attributes for the user are specified as a JavaScript associative array indexed by attribute name. The *objectClasses* parameter specifies any additional object classes as a JavaScript array. If the addition of the user is successful, returns `true`, otherwise returns `false`.

• `userDN`	A String of the distinguished name (DN) for the new user entry.
• `attributes`	An Object that is a table of key/value pairs specifying user attributes.
• `objectClasses`	An Array of object classes with which the user entry is compliant.

Example:

```
var attrs = new Object();
attrs[ "uid" ] = "joe";
attrs[ "cn" ] = "Joe Cool";
attrs[ "sn" ] = "Cool";
attrs[ "mail" ] = "joe@acme.com";
attrs[ "favoriteColor" ] = "green";
// Specify additional objectclasses
var OCs = new Array( "favoriteColorOC" );
// add the new user
cd.addUser( "uid=joe, ou=People, o=acme.com", attrs, OCs );
```

deleteUserByCN

`deleteUserByCN(userCN)`

Deletes the user entry specified by the Common Name (CN) *userCN* from the corporate directory. Returns `true` if the deletion is successful. Returns `false` if the deletion failed.

• `userCN`	A String of the common name (CN) for the user entry to be deleted.

Example:

```
var cd = getCorporateDirectory();
var joe = cd.getUserById( "joe" );
// Delete the user
cd.deleteUserByCN( joe.cn );
```

deleteUserByDN

`deleteUserByDN(userDN)`

Deletes the user entry specified by the Distinguished Name (DN) *userDN* from the corporate directory. Returns `true` if the deletion is successful. Returns `false` if the deletion failed.

- `userDN` A String of the distinguished name (DN) for the user entry to be deleted.

Example:

```
var cd = getCorporateDirectory();

var joe = cd.getUserById( "joe" );

cd.deleteUserByDN( joe.getDN );
```

deleteUserById

`deleteUserById(uid)`

Deletes the user entry specified by userID from the corporate directory. Returns `true` if the deletion is successful. Returns `false` if the deletion failed.

- `userDN` A String of the DN for the user entry to be deleted

Example:

```
var cd = getCorporateDirectory();
var joe = cd.getUserById( "joe" );
cd.deleteUserByCN( joe.getUserId());
```

getUserByCN

`getUserByCN(userCN)`

Returns the user entry corresponding to the Common Name (CN) *userCN* as a JavaScript User object. User object is a Process Manager-specific object. Returns `null` if no user entry corresponds to *userCN*.

• `userCN`	A String of the common name (CN) of the user entry to be retrieved.

Example:

```
var cd = getCorporateDirectory();
var joe = cd.getUserByCN( "Joe Cool" );
```

getUserByDN

`getUserByDN (userDN)`

Returns the user entry corresponding to the Distinguished Name (DN) *userDN* as a JavaScript User object. User object is a Process Manager-specific object. Returns `null` if no user entry corresponds to *userDN*.

• `userDN`	A String of the distinguished name (DN) of the user entry to be retrieved.

Example:

```
var cd = getCorporateDirectory();

var joe = cd.getUserByDN( "uid=joe, ou=People, o=acme.com" );
```

getUserById

`getUserById (uid)`

Returns the user entry corresponding to the user id *userId* as a JavaScript User object. The User object is a Process Manager-specific object. Returns `null` if no user entry corresponds to *userId*.

• `uid`	A String of the User ID for the user entry to be retrieved.

Example:

```
var cd = getCorporateDirectory();
var joe = cd.getUserByID( "joe" );
```

modifyUserByCN

`modifyUserByCN (userCN, attrName, attrValue, operation)`

• `userCN`	A String of the common name (CN) of the user entry to be modified.
• `attrName`	A String of the name of the user entry attribute to modify.
• `attrValue`	A String of the value of the user entry attribute to modify.
• `operation`	A String of the modification operation to be performed.

The type of modification is specified by the operation, which can have the following values:

ADD: adds the attribute attrName with the value attrValue to the user entry. If attrName has multiple values, the operation can result in multiple instances of attrName in the user entry. attrName should be a valid attribute of one of the object classes with which the user entry complies.

REPLACE: replaces the value currently associated with attrName with the new value attrValue.

DELETE: deletes attrName from the user entry.

Example:

```
var cd = getCorporateDirectory();
cd.modifyUserByCN( "Nikki Beckwell",
  "favoriteColor", "blue", "Add");
```

modifyUserByDN

`modifyUserByDN (userDN, attrName, attrValue, operation)`

Modifies the user entry specified by the Distinguished Name (DN) *userDN* in the corporate directory. Returns true if the modification is successful. Returns false if the modification fails.

• `userDN`	A String of the distinguished name (DN) of the user entry to modify.
• `attrName`	A String of the name of the user entry attribute to modify.
• `attrValue`	A String of the value of the user entry attribute to modify.
• `operation`	A String of the modification operation to perform.

The type of modification is specified by the operation, which can have the following values:

ADD: adds the attribute attrName with the value attrValue to the user entry. If attrName has multiple values, the operation can result in multiple instances of attrName in the user entry. attrName should be a valid attribute of one of the object classes with which the user entry complies.

REPLACE: replaces the value currently associated with attrName with the new value attrValue.

DELETE: deletes attrName from the user entry.

Example:

```
var cd = getCorporateDirectory();
cd.modifyUserByDN( "uid=joe, ou=People, o=acme.com",
  "favoriteColor", "blue", "ADD" );
cd.modifyUserByDN( "uid=carol, ou=People, o=acme.com",
  "favoriteColor", "red", "REPLACE" );
cd.modifyUserByDN( "uid=barbara, ou=People, o=acme.com",
  "favoriteColor", "", "DELETE" );
```

modifyUserById

`modifyUserById (userID)`

• `userID`	String	A String of the User ID of the user entry to modify.
• `attrName`	String	A String of the name of the user entry attribute to modify.
• `attrValue`	String	A String of the value of the user entry attribute to modify
• `operation`	String	A String of the modification operation to perform.

The type of modification is specified by the operation, which can have the following values:

ADD: adds the attribute attrName with the value attrValue to the user entry. If attrName has multiple values, the operation can result in multiple instances of attrName in the user entry. attrName should be a valid attribute of one of the object classes with which the user entry complies.

REPLACE: replaces the value currently associated with attrName with the new value attrValue.

DELETE: deletes attrName from the user entry.

Example:

```
var cd = getCorporateDirectory();
cd.modifyUserByCN( "nbeckwell","favoriteColor", "blue", "ADD" );
```

User

The `User` object has publicly accessible methods and exposes the user attributes making up an LDAP user entry. You can treat the `User` object as a hashtable of attribute key/value pairs where the keys are the attribute names as they appear in the LDAP entry. In the following code sample, you can also access the attribute `favoritecolor`, which is an attribute available from the user `joe`'s entry.

Example:

```
var cd = getCorporateDirectory();
var u = cd.getUserById( "joe" );
var email = u.mail;
var favColor = u.favoritecolor;
```

The following methods can be called on the `User` object:

- getUserId

getUserId

`getUserId()`

Returns the user ID for the current user. The user ID attribute can also be accessed directly as the `uid` property of the user object.

Example:

```
var cd = getCorporateDirectory();
var u1 = cd.getUserByDN( "uid=joe, ou=People, o=acme.com" );
var id1 = u1.getUserID();

var pi = getProcessInstance();
var u2 = pi.getCreatorUser();
var id2 = u2.getUserId() );
```

Logging and Error Handling Global Functions

The following global functions are available for adding entries to error, informational, and history logs.

- logErrorMsg
- logHistoryMsg
- logInfoMsg
- logSecurityMsg

logErrorMsg

`logErrorMsg (label, context)`

Adds an entry in the error log of the application. This information is displayed to the administrator when viewing the error log.

• `label`	String of the label for the error log's entry.
• `context`	(Optional) Object that is a Hashtable of key/value pairs to print in the log entry.

The following code could be used in a completion script or automation script to log an error message when a user tries to submit a document for review when the document is too long.

Example:

```
// LOG AN ERROR MESSAGE
var pi = getProcessInstance():
var author = pi.getData("author");
var docName = pi.getData("docName");
var pageCount = parseInt (pi.getData("pageCount"));

if (pageCount > 5000) {
  var error1 = new Object ();
  error1.author = author;
  error1.docName = docName;
  error1.pageCount = pageCount;
  logErrorMsg ("PAGE_COUNT_TOO_LONG", error1);
}
```

logHistoryMsg

`logHistoryMsg (label, comment)`

Adds a row in the history log. This information is displayed to users when viewing Details & History in a work item list.

• `label`	String of the label for the entry in the history log.
• `comment`	(Optional) String of an additional comment for the history event.

The *label* is a string to be used as the label for the entry in the history log, and the optional argument *comment* is a string describing the entry.

Use this function to add information for the users about the work items. The following code could be used in a completion script to add details when an author submits a document for review.

Example:

```
// LOG A HISTORY MESSAGE
var pi = getProcessInstance():
var author = pi.getData("author");
var docName = pi.getData("docName");
var pageCount = parseInt (pi.getData("pageCount"));

logHistoryMsg ("Doc Submitted for Review",
  " \nAuthor: " + author +
  " \nDocument name: " + docName +
  " \nPage count: " + pageCount);
```

logInfoMsg

`logInfoMsg(label, context)`

Adds an entry in the info log of the application. This information is displayed to the administrator when they view the info log.

• `label`	String of the label for the entry in the history log.
• `context`	(Optional) Object that is a hashtable of key/value pairs to print in the log entry.

The *label* is a string to be used as the label for the entry in the info log, and the optional argument *context* is an object that has a variable for each property of the info message.

Use this function to show information about the execution of scripts. This is particularly useful for viewing the progress of script execution while debugging scripts. The following code could be used in a completion or automation script to log information about a document that has been submitted for review.

Example:

```
// LOG AN INFO MESSAGE
var pi = getProcessInstance():
var author = pi.getData("author");
var docName = pi.getData("docName");
var pageCount = parseInt (pi.getData("pageCount"));

var info1 = new Object ();
info1.author = author;
info1.docName = docName;

logInfoMsg ("DOC_SUBMITTED_FOR_REVIEW", info1);
```

logSecurityMsg

`logSecurityMsg (label, context)`

Adds an entry in the security log of the application. This information is displayed to the administrator when they view the security log.

• `label`	String of the label for the entry in the security log.
• `context`	(Optional) Object that is a hashtable of key/value pairs to print in the log entry.

The *label* is a string to be used as the label for entry in the security log, and the optional argument *context* is an object that has a variable for each property of the error message.

The following code could be used in a completion script or automation script to log a warning when an unauthorized user tries to perform a task that accesses an external database.

Example:

```
// LOG A SECURITY MESSAGE
var pi = getProcessInstance():
var username = pi.getData("username");
var password = pi.getData("password");
var externalDB = pi.getData("externalDBName");

var securityError1 = new Object ();
securityError1.username = username;
securityError1.password = password;
securityError1.externalDB = externalDB;

logSecurityMsg ("EXTERNAL_DB_ACCESS_NOT_ALLOWED",
securityError1);
```

Assignment, Completion, and Email Scripts

This section lists the predefined assignment, completion, and email scripts available in Process Automation Edition. These predefined scripts (which are also global functions) can be called from other scripts. For example, an expiration handler script could call the predefined function `toUserById()`. The value returned by `toUserById()` could in turn be passed to the `assignTo()` method on a work item. The `assignTo()` method would then re-assign the work item to a user with a given user ID.

The global functions in this section are:

- checkParallelApproval
- defaultNotificationHeader
- defaultNotificationSubject
- emailById
- emailOfAssignees
- emailOfCreator
- emailOfRole
- randomToGroup
- toCreator
- toGroup
- toManagerOf
- toManagerOfCreator
- toManagerOfRole
- toParallelApproval
- toUserById
- toUserFromField

checkParallelApproval

`checkParallelApproval (dataField, stopAction)`

This function is meant for use as a completion script for work items that use the `toParallelApproval()` assignment script. The *dataField* must be the name of the data field that was specified in `toParallelApproval()`, and is used to keep track of who has completed the work item so far. The *stopAction* is the name of the action that a user can take to stop the approval process.

• `dataField`	String of the name of a data field.
• stopAction	String of the name of an action.

If an activity (work item) uses the `toParallelApproval()` assignment script, the activity should have two possible actions for the assignees to take when they complete the work item. One should indicate approval, and the other should indicate disapproval. The activity should also use the `checkParallelApproval()` completion script, which performs the "disapproval" action as soon as any one assignee selects it, the theory being that it takes everyone to approve the work item, but only one dissenter to disapprove it.

Example:

```
checkParallelApproval( "trackerField", "Reject" );
```

defaultNotificationHeader

`defaultNotificationHeader()`

Returns the default notification header for a notification message body. The header contains information about the current work item, such as the current activity name, the process instance ID, the creation date of the process instance and the expiration date (if any).

This function may be used as the notification body script by itself, or may be embedded in your own notification body script. You may also use this function from a template evaluated using `evaluateTemplate()`. The function may only be used successfully from a script associated with a notification; if used anywhere else, an empty string is returned.

The text returned from this function will depend upon the content-type of the notification. If the content-type is `text/html`, the header will be a series of HTML tags; if the content-type is `text/plain`, the header will be plain text.

Example:

```
function emailBody( )
{
var body = "Email message body <br>";
body += defaultNotificationHeader();
return body;
}
```

defaultNotificationSubject

`defaultNotificationSubject()`

Returns the default notification subject for the notification subject line. The subject contains information about the current process instance, such as the process instance ID the priority and the title string.

This function may be used as the notification subject script by itself, or may be embedded in your own notification subject script. The function may only be used successfully from a script associated with a notification; if used anywhere else, an empty string is returned.

Example:

```
function emailSubject( )
{
var subject = "Subject: " + defaultNotificationSubject();
return subject;
}
```

emailById

`emailById(userId)`

Returns a string of comma-delimited email addresses for the user with the given user ID. The `mail` attribute for the user must contain a valid email address in the corporate user directory. If the `mail` attribute of the user does not have a value, this function logs an error and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

• `userId`	String of the User Id whose email address is to be returned.

Example:

```
emailById( "joe" );
```

emailOfAssignees

`emailOfAssignees()`

Returns a string of comma-delimited email addresses for all the assignees of the work item. The `mail` attribute for each assignee must contain a valid email address in the corporate user directory. If the `mail` attribute is empty for any assignee, no address is added to the string for that assignee. If no assignee has a value in their mail attribute, the function logs an error message and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

Example:

```
emailOfAssignees();
```

emailOfCreator

`emailOfCreator()`

Returns a string of the email address of the user who created the process instance. The user's `mail` attribute must contain a valid email address in the corporate user directory. If the `mail` attribute of the user does not have a value, this function logs an error and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

Example:

```
var pi = getProcessInstance();
var creatorEmail = emailOfCreator();
pi.setData( "creatorEmail", creatorEmail );
```

emailOfRole

`emailOfRole(roleName)`

Returns a string of the email address of the user performing the given role. The user's `mail` attribute must contain a valid email address in the corporate user directory. If the `mail` attribute of the user does not have a value, this function logs an error and returns `null`. This function is intended for use as a notification script, but can be used anywhere that a string of email addresses is needed.

• `roleName`	String of a role name

Example:

```
emailOfRole( "reviewer" );
```

randomToGroup

`randomToGroup(groupName)`

This function returns an array containing a user ID of one member of the group. The group member is selected at random. This function is intended to be used as an assignment script but can be used anywhere that an array of user IDs is needed.

• `groupName`	String of the name of a group.

Example:

```
var userArray = randomToGroup( "helpDesk" );
```

toCreator

`toCreator()`

Returns a JavaScript array with the user ID of the user who created the process instance. This method is the default assignment script assigned to user activities when a node is created on the Process Map.

Example:

```
var creatorArray = toCreator();
var creator = creatorArray[ 0 ]);
```

toGroup

`toGroup(groupName)`

Returns the qualified role name of the group. If the parameter *groupName* does not correspond to an actual group in the application's Groups and Roles folder, returns `null`.

- `groupName` String of the name of the group.

toManagerOf

`toManagerOf (userId)`

This function returns an array containing the user ID of the manager of the user with the given user ID. The `manager` attribute of the given user must contain a DN in the corporate user directory. This function is intended to be used as an assignment script but can be used anywhere that an array of user IDs is needed.

- `userId` A String of a user ID.

Example:

```
var managerArray = toManagerOf( "mike" );
var manager = managerArray[0];
```

toManagerOfCreator

`toManagerOfCreator()`

This function returns an array containing the user ID of the manager of the creator of the process instance. The `manager` attribute of the creator user must contain a DN in the corporate user directory. This function is intended to be used as an assignment script but can be used anywhere that an array of user IDs is needed.

Example:

```
var managerArray = toManagerOfCreator();
var creatorManager = managerArray[ 0 ];
```

toManagerOfRole

`toManagerOfRole(role)`

This function returns an array containing the user ID of the manager of the user defined as the given role. The `manager` attribute of the user fulfilling the role must contain a DN in the corporate user directory. This function is intended to be used as an assignment script but can be used anywhere that an array of user IDs is needed.

• `role`	String of a name of the role.

Example:

```
var managerArray = toManagerOfRole( "reviewer" );
var reviewerManager = managerArray[ 0 ];
```

toParallelApproval

`toParallelApproval(arrayOfUserDNs, dataField)`

This function should be used as an assignment script to assign a work item to several users who must all complete the work item. The *arrayOfUsers* argument is an array of user IDs, and *dataField* is a data field that keeps track of who has performed the work item and who still needs to do it. This field is a computed field of length 2000 that must add to the data dictionary.

• `arrayOfUsers`	Array of user IDs.
• dataField	String of a name of a data field in the current process.

If an activity (work item) uses the `toParallelApproval()` assignment script, the activity should have two possible actions for the assignees to take when they complete the work item. One action should indicate approval, and the other should indicate disapproval. The activity should also use the `checkParallelApproval()` completion script, which performs the “disapproval” action as soon as any one assignee selects it. The idea is that it takes everyone to approve the work item, but only one dissenter to disapprove it.

Example:

```
var cd = getCorporateDirectory();
var user1 = cd.getUserById( "user1" );
var user2 = cd.getUserById( "user2" );
var user3 = cd.getUserById( "user3" );
var approvers = new Array( user1, user2, user3);
toParallelApproval( approvers, "trackerField" );
toParallelApproval(toGroup( "approvers" ), "trackerField" );
```

toUserById

`toUserById(userId)`

This function returns an array containing the user ID of the user with the given user ID. This function is intended to be used as an assignment script but can be used anywhere that an array of user IDs is needed.

• `userId`	String of a user ID.

Example:

```
var userArray = toUserById( "sijacic" );
```

toUserFromField

`toUserFromField(dataField)`

This function returns an array containing the user IDs of the user whose user ID is the value of the given data field. This function is intended to be used as an assignment script but can be used anywhere that an array of user IDs is needed.

`dataField`	String of a name of a data field whose value is a user ID.

Example:

```
var userArray = toUserFromField( "approver" );
var approver = userArray[0];
```

Miscellaneous Global Functions

This section summarizes the global functions that are neither log and error functions nor predefined scripts.

The global functions in this section are:

- `ejbLookup`
- `evaluateTemplate`
- `expireIn`
- `getAction`
- `getApplicationName`
- `getApplicationPath`
- `getApplicationPrettyName`
- `getBaseForFileName`
- `getConnector`
- `getContentStore ()`
- `getCreatorUserId`
- `getCorporateDirectory`
- `__getIncludePath`
- `getJndiNamingContext`
- `getProcessInstance`
- `getSubProcessInstance`
- `getWorkItem`
- `__includeFile`
- `mapTo`
- `mount`
- `setConnector`
- `__setIncludePath`
- `setRedirectionURL`
- `url_OnDisplayHistory`
- `url_OnDisplayProcessInstance`
- `url_OnDisplayWorklist`
- `url_OnListApplications`
- `url_OnListEntryNodes`

ejbLookup

`ejbLookup(jndiName)`

Looks up an EJB that can be found under the given JNDI name. If no EJB exists under this name, an EJB exception, `javax.naming.NamingException`, is thrown. Process Manager handles this exception by moving the process instance into the designated Exception Manager node.

If the bean can be found under the given JNDI name, the home interface of that bean is returned. The home interface is used to create new instance of the EJB or find existing instances of the given EJB. The definition of the home interface is completely application-dependent; it typically consists of several `create()` and `finder()` methods. This function call is equivalent to the `getJndiNamingContext().lookup(jndiName)` function call.

• `jndiName`	A String of the JNDI name of the EJB to look up.

Example:

```
var home = ejbLookup("iPlanet/CreditCardServer");
// Create a credit card server that can do card authorizations.
var creditCardServer = home.create ( "mastercard" );
```

evaluateTemplate

`evaluateTemplate(templateName)`

Used primarily in email notifications, but can be used for any template-like evaluation.

For email notifications, use `evaluateTemplate()` in the email body. When you create an email notification in Process Builder, specify `evaluateTemplate( "templateFile" )` as the body script, where `templateFile` is the name of the template file relative to the application directory on the server.

Include your template files in the following directory on the local machine where Process Builder resides: <*iPM_install_directory*>`/builder/Applications/`<*your_app*>`/templates`. When you deploy your application, the same directory structure is replicated on the application server.

`evaluateTemplate()` also evaluates JavaScript segments contained in the template file. These segments are identified by `<script language="Rhino">` tags.

In an HTML page containing client-side JavaScript, you can replace

`<script language="JavaScript">`

with

`<script language="Rhino">`

The Process Manager engine will evaluate the JavaScript in the same manner the browser does.

You can access the entire engine API (e.g., `getProcessInstance()`, `getWorkITem()`, etc.) from within these segments. To send output to an HTML file, use `document.write()` with the output you want to send provided as an argument.

`evaluateTemplate()` returns a string. The string is made up of the contents of the template file with the Rhino segments evaluated. If there are no Rhino segments in the file, the string simply contains the file contents. If there are Rhino segments, the string contains the evaluated JavaScript rather than the Rhino segments.

Example:

```
var fileContents = evaluateTemplate( "templateFile" );
var cs = getContentStore():
cs.upload( fileContents, url ):
```

expireIn

`expireIn(val, unit)`

Returns a JavaScript `Date` object set to the absolute `datetime` when the current work item expires.

- `val` An Integer specifying the number of units in which to expire the workitem.
- `unit` A String that specifies the unit of time measurement, such as minutes or hours.

This function is normally used to specify the expiration time for a particular activity, and is usually set by the designer through Process Builder. In the event that you choose to specify the arguments for this function manually, the parameter unit can contain the following values:

- minutes
- hours
- days
- weeks
- months

Note that a month is presumed to contain 30 days. Asking a work item to expire one month from the date January 1st results in an expiration date of January 30th, not February 1st.

Example:

```
expireIn( 6, "minutes" );    // expire in 6 minutes
expireIn( 30, "days" );     // expire in 30 days
expireIn( 1, "month" );     // expire in 30 days
```

getAction

`getAction()`

Returns the name of a transition (button). Examples of possible return values for `getAction` are “`deny`” or “`approve`”.

The transition corresponds to the name of the button clicked when a work item form is submitted. This function can be used in any script, but is best used from the completion script of a user activity.

Example:

```
function completion( )
{
  var pi = getProcessInstance();
  pi.setData( "submittedAction", getAction() ):
  return true;
}
```

getApplicationName

`getApplicationName()`

Returns the name of the application (for example: `"DataSheet"`) as a string.

Example:

```
var appName = getApplicationName();
```

getApplicationPath

`getApplicationPath()`

Returns the pathname of the directory containing the files for this application as a string.

Example:

```
var appPath = getApplicationPath();
```

getApplicationPrettyName

`getApplicationPrettyName()`

Returns the pretty name of the application (for example: `"DataSheet Management"`) as a string.

Example:

```
var appPN = getApplicationPrettyName();
```

getBaseForFileName

`getBaseForFileName (processId)`

Given a unique process instance ID, this function returns a string of the base file name for the content store for that process instance.

• `processId`	A String of the process instance ID.

The following code creates a path name for the file `myFile.html` in the content store. Note that this code does NOT create the file, it simply creates a string that can be used as the file name by methods on a contentStore object, such as `move()` and `store()`.

Example:

```
var pi = getProcessInstance();
var pid = pi.getInstanceId();
var contentStorePath = getBaseForFileName (pid);
var newFileName = contentStorePath + "myFile.html";
```

getConnector

`getConnector(connectorKey)`

Given a connector key, returns a connector object from the list of connector objects available for use by scripts.

• `connectorKey`	A String specifying a connector ID (previously set by `setConnector()`).

Example:

```
function initialisationScript( )
{
  var c = new Packages.com.acme.DBConnection();
  setConnector( "dbConnector" );
  return true;
}
var c = getConnector( "dbConnector" );
```

getContentStore ()

`getContentStore ()`

`getContentStore(httpURL,user,password)`

Returns a `contentStore` object connected to the content store. If no arguments are passed, this method gets the content store associated with the current application.

If `httpURL`, `user` and `password` arguments are provided, this method returns a content store object connected to the given URL.

You must have a content store object to use any of the content store methods.

• `httpURL`	A String of the URL of an HTTP server capable of acting as a content store.
• `user`	A String of the User ID of the user who will authenticate the content store specified by `httpURL`.
• `password`	Password for the specified user.

Example:

```
var cs = getContentStore();

var publishCS = getContentStore(
"http://publish.iplanet.com/CS", "sijacic", "password");
```

getCreatorUserId

`getCreatorUserId ()`

Returns the user ID for the creator user of the current process instance.

This function is intended to be used as the `initiate-as` script for sub-process nodes. When the child process instance is created, its creator must be determined. Normally, the creator of the child process instance is the same user that authenticated with the server at the time the sub-process node was executed. However, it may be desirable (or necessary) to set the child creator to the same user as the creator of the parent process instance.

If you choose to use this function to change the child creator user, the authenticating (or current) user must be a member of the trusted users group in the child application.

Example:

```
var pi = getProcessInstance();
var creatorUser = pi.getCreatorUser();
var creatorUserId = creatorUser.getUserId();
```

getCorporateDirectory

`getCorporateDirectory ()`

Returns a `corporateDirectory` object connected to the corporate directory in the Directory Server. See the section "CorporateDirectory" for details of the methods on this object.

Example:

```
var corpDir = getCorporateDirectory();
```

__getIncludePath

`__getIncludePath ()`

Returns the current inclusion path used by the server to search for JavaScript source files. This search path resembles in behavior the PATH mechanism used in Windows NT and UNIX to search for executables and libraries when a file name is not absolute.

The directory *installDir*`/resources/server/js` (where *installDir* is Process Manager installation directory on the server machine) is included in the search path by default.

Example:

```
// Returns the current search path
var includePath = __getIncludePath();

// Add our own directory to the search path
includePath += ';' + "d:\\tmp";

// Set the search path
__setIncludePath( includePath );
```

getJndiNamingContext

`getJndiNamingContext()`

Returns the current iAS JNDI naming context used to look up the home interface of an EJB.

Use this function to access EJBs from within a Custom Activity. You can pass the instance of the naming context to the Custom Activity and use the `lookup()` method on this context to obtain home interfaces of the beans.

Example:

```
var namingContext = getJndiNamingContext()
```

getProcessInstance

`getProcessInstance ()`

Returns a JavaScript ProcessInstance object associated with the current work item.

See the section ProcessInstance for details of the methods on this object.

Example:

```
var wi = getWorkItem();
var pi = getProcessInstance();
var pid = pi.getInstanceId();
```

getSubProcessInstance

`getSubProcessInstance ()`

Returns a ProcessInstance object corresponding to the completed child process instance. This function should only be used from the completion script of a sub-process node. The object returned by this function has all the same methods expected of a ProcessInstance object, except that the information contained within corresponds to the completed child process instance instead of the parent. If this function is called from any location other than the completion script of a sub-process node, `null` is returned.

Example:

```
// Completion script of sub-process node
function parentCompletion( )
{
var pi = getProcessInstance();        // parent process instance
var spi = getSubProcessInstance();    // child process instance

// Map some of the data elements from the child back to the
// parent process instance
pi.setData( "childStatus", spi.getData( "status" ) );
pi.setData( "numApprovers", spi.getData( "numApprovers" ) );
return true;
}
```

getWorkItem

`getWorkItem ()`

Returns a `workItem` object that represents the current work item. See the section WorkItem for details of the methods on this object.

Example:

```
var wi = getWorkItem();
```

__includeFile

`__includeFile (fileName)`

Reads in the contents of a JavaScript source file and evaluates the contents. Any JavaScript functions and objects defined within the file are available to all user scripts.

The function `__includeFile()` and its “helper” functions, `__getIncludePath()` and `__setIncludePath()`, are used by the server to define the built-in JavaScript functions available to user scripts.

If you want to access JavaScript functions or objects from user scripts and do not want to include the script files with every application you develop, you can externalize the functions in a separate JavaScript file and include the functions in the server's global run-time scope using `__includeFile()`.

• `fileName`	A String of the pathname of the JavaScript file to evaluate.

Example 1:

```
// Sample JavaScript file
function checkCreator( userId )
{
  var pi = getProcessInstance();
  var u = pi.getCreatorUser();

  // If the process instance creator matches the
  // userId, return true, else false.
  if( userId == u.getUserId() )
      return true;
  else
      return false;
}
```

Example 2

```
// Sample includeList.js file
var myIncludePath = "d:\\tmp";
var includePath = __getIncludePath() + ';' + myIncludePath;

__setIncludePath( includePath );
__includeFile( "myFunctions.js" );
```

mapTo

`mapTo(fieldName)`

Used only from a custom activity as a means of mapping values from the output hashtable back to the process instance. The *fieldName* parameter specifies the name of a data element in the process instance. This data element receives the value of the current output parameter. When the custom activity's `perform()` call is complete, the server iterates through the hashtable, one element at a time, and attempts to map the element's value to a field in the process instance. Returns `true` if data mapping succeeds. Returns `false` if a problem has occurred.

• `fieldName`	A String of the name of a data element in the process instance.

Example:

```
mapTo( "customerName" );
```

mount

`mount(jndiName)`

Looks up and creates an instance of a stateless session bean, or EJB, found under the given JNDI name. If no bean exists under this name, a `javax.naming.NamingException` is thrown. Process Manager handles this exception by moving the process instance into the designated Exception Manager node. If the EJB can be found under the given JNDI name, you create an instance of it by calling the `create()` method on the returned home interface of the bean. This function call is equivalent to the `ejbLookup(jndiName).create()` function call.

• `jndiName`	A String of the JNDI name of the EJB to "mount."

Example:

```
var creditServer = mount("iPlanet/CreditCardServer");
```

setConnector

`setConnector(connKey, connObject)`

Adds a connector object (such as a database connection) indexed by key to the list of connector objects that can be used by scripts.

• `connKey`	A String of the key used to retrieve the associated object (through `getConnector()`).
• connObject	Object stored and associated with the specified key.

JavaScript objects saved inside the Connector, such as those created using `var obj = new Object ( )`, are converted into Java objects based on the mapping table below before being stored in the connector

JavaScript Data Type	**Converted Java Data Type**
• string	java.lang.String
• number	java.lang.Double
• boolean	java.lang.Boolean
• date	java.util.Date
• array	java.lang.Object[]
• object	java.util.Hashtable
• function	java.lang.String (decompiled source)
• script	java.lang.String (decompiled source)
• java.lang.Object (wrapped)	java.lang.Object (unwrapped)
• java.lang.Object	java.lang.Object (pass-through)

Example:

```
function initialisationScript( )
{
var c = new Packages.com.acme.DBConnection();
setConnector( "dbConnector", c );
return true;
}

// This DB connection can now be used in any script.
var c = getConnector( "dbConnector" );
```

__setIncludePath

`__setIncludePath (includePath)`

Allows the user to specify the path the server uses to search for JavaScript source files. The path is a delimited list of absolute directory path names. The delimiter depends upon the system:

- : (a colon) for UNIX
- ; (a semicolon) for Windows

Searches start from the first directory and proceed to the last directory until the file is located. The directory *installDir*`/resources/server/js` (where *installDir* is where Process Manager is installed on the server machine) is included in the search path by default. However, once you set your own path using this function, the current search path is overwritten. Thus, calls to `__setIncludePath` should specify the entire search path by appending the new search directory to the current search path.

`includePath`	A String of the search path the server uses to locate JavaScript source files.

Example:

```
var includePath = __getIncludePath();
includePath += ';' + "d:\\tmp";

// Set the search path.  Note how the variable includePath also
// specifies the existing search path.
// Every call to __setIncludePath
// overwrites the previous search path.
__setIncludePath( includePath );
```

setRedirectionURL

`setRedirectionURL(stringURL)`

Redirects the participant's browser to the specified location in `stringURL`.

Invoke this method from a completion script so that when the activity has completed, the user immediately sees another HTML page such as a form rather than the standard activity completion dialog box.

The `setRedirectionURL()` method is typically used for multiple-screen data entry where the same person is required to enter data across several screens of user activities. The process subsequently becomes a wizard-like forms entry screen.

• `stringURL`	A String of a valid URL.

Example:

```
function scriptComplete( )
{
  // Redirect to the Sun Home Page
  setRedirectionURL( "http://www.sun.com" );

  // Alternatively, this causes the next work item in the same
  // process instance to be displayed after
  // the user clicks an action button
  setRedirectionURL( url_OnDisplayProcessInstance() );

  // Don’t forget to return true;
  // this is still a completion script
  return true;
}
```

url_OnDisplayHistory

`url_OnDisplayHistory()`

Returns a string containing the URL that points to the history list for the current process instance. This function is usually used in conjunction with `setRedirectionURL()` to redirect a user to the history page of the current process instance.

Example:

```
function scriptComplete( ) {
// Redirect to the History Page of this process instance
setRedirectionURL( url_OnDisplayHistory() );
// Don’t forget to return true;
// this is still a completion script
return true;

}
```

url_OnDisplayProcessInstance

`url_OnDisplayProcessInstance()`

Returns a string containing the URL that points the current process instance. This function is usually used in conjunction with `setRedirectionURL()` to redirect a user to the current process instance.

Example:

```
function scriptComplete( )
{
// Redirect to the current process instance
setRedirectionURL( url_OnDisplayProcessInstance() );
// Don't forget to return true;
// this is still a completion script
return true;

}
```

url_OnDisplayWorklist

`url_OnDisplayWorklist()`

`url_OnDisplayWorklist()`

Returns a string containing the URL that points to the user's work list. This function is usually used in conjunction with `setRedirectionURL()` to redirect a user to their work list.

Example:

```
function scriptComplete( )
{
// Redirect to the user to their work list
setRedirectionURL( url_OnDisplayWorklist() );
// Don't forget to return true;
// this is still a completion script
return true;

}
```

url_OnListApplications

`url_OnListApplications()`

Returns a string containing the URL listing applications installed on the Process Manager cluster. This function is usually used in conjunction with `setRedirectionURL()` to redirect a user to the list of applications installed on the cluster.

Example:

```
function scriptComplete( )
{
// Redirect to the list of applications on this cluster
setRedirectionURL( url_OnListApplications() );
// Don’t forget to return true;
// this is still a completion script
return true;

}
```

url_OnListEntryNodes

`url_OnListEntryNodes()`

Returns a string containing the URL that points to the list of entry points in the current application. This function is usually used in conjunction with `setRedirectionURL()` to redirect a user to the list of entry points in the current application.

Example:

```
function scriptComplete( )
{
// Redirect to the list of entry points of the current application
setRedirectionURL( url_OnListEntryNodes() );
// Don't forget to return true;
// this is still a completion script
return true;

}
```

Alphabetical Summary of JavaScript Methods and Functions

The following table lists the JavaScript methods and functions available in Process Manager:

JavaScript Method or Function		Where Used in Process Manager
__getIncludePath	page 441	global function
__includeFile	page 443	global function
__setIncludePath	page 447	global function
addGroupAssignee	page 393	workItem
addUser	page 413	corporateDirectory
addUserAssignee	page 392	workItem
assignees	page 393	workItem
checkParallelApproval	page 425	predefined script
copy	page 399	contentStore
defaultNotificationHeader	page 425	predefined script
defaultNotificationSubject	page 426	predefined script
deleteUserByCN	page 413	corporateDirectory
deleteUserByDN	page 414	corporateDirectory
deleteUserById	page 414	corporateDirectory

JavaScript Method or Function		Where Used in Process Manager
download	page 400	contentStore
ejbLookup	page 433	global function
emailById	page 427	predefined script
emailOfAssignees	page 427	predefined script
emailOfCreator	page 427	predefined script
emailOfRole	page 428	predefined script
evaluateTemplate	page 434	global function
exists	page 400	contentStore
expireIn	page 435	global function
extend	page 393	workItem
getAction	page 436	workItem
getApplicationName	page 437	global function
getApplicationPath	page 437	global function
getApplicationPrettyName	page 438	global function
getBaseForFileName	page 438	global function
getBaseURL	page 401	global function
getConnector	page 439	global function
getContent	page 402	contentStore
getContentStore ()	page 439	global function
getCorporateDirectory	page 440	global function
getCreationDate	page 386	processInstance
getCreationDate	page 394	workItem
getCreatorUser	page 386	processInstance
getCreatorUserId	page 440	global function
getData	page 387	processInstance
getEntityKey	page 387	processInstance
getEntryNodeName	page 388	processInstance
getException	page 403	contentStore
getExitNodeName	page 388	processInstance

JavaScript Method or Function		Where Used in Process Manager
getExpirationDate	page 394	workItem
getInstanceId	page 388	processInstance
getJndiNamingContext	page 441	processInstance
getNodeName	page 394	workItem
getPriority	page 389	global function
getProcessInstance	page 442	processInstance
getRoleUser	page 389	contentStore
getRootURL	page 404	contentStore
getSize	page 404	contentStore
getStatus	page 405	contentStore
getSubProcessInstance	page 442	global function
getTitle	page 389	processInstance
getUserByCN	page 415	corporateDirectory
getUserByDN	page 415	corporateDirectory
getUserById	page 416	corporateDirectory
getUserId	page 420	user
getVersion	page 405	contentStore
getWorkItem	page 443	global function
hasExpired	page 395	workItem
initialize	page 406	contentStore
isException	page 407	contentStore
isStateActive	page 395	workItem
isStateRunning	page 395	workItem
isStateSuspended	page 395	workItem
list	page 407	contentStore
logErrorMsg	page 420	global function
logHistoryMsg	page 421	global function
logInfoMsg	page 422	global function
logSecurityMsg	page 423	global function

JavaScript Method or Function		Where Used in Process Manager
mapTo	page 445	global function
mkdir	page 408	contentStore
modifyUserByCN	page 416	corporateDirectory
modifyUserByDN	page 417	corporateDirectory
modifyUserById	page 418	corporateDirectory
mount	page 445	global function
move	page 408	contentStore
moveTo	page 396	workItem
randomToGroup	page 428	predefined script
remove	page 409	contentStore
removeAssignees	page 397	workItem
resume	page 397	workItem
rmdir	page 410	contentStore
setConnector	page 446	global function
setData	page 390	processInstance
setEntityKey	page 391	global function
setExpirationDate	page 397	workItem
setRedirectionURL	page 448	global function
setRoleById	page 391	processInstance
store	page 411	contentStore
suspend	page 398	workItem
toCreator	page 429	predefined script
toGroup	page 429	predefined script
toManagerOf	page 429	predefined script
toManagerOfCreator	page 430	predefined script
toManagerOfRole	page 430	predefined script
toParallelApproval	page 431	predefined script
toUserById	page 432	predefined script
toUserFromField	page 432	predefined script

JavaScript Method or Function		Where Used in Process Manager
upload	page 412	contentStore
url_OnDisplayHistory	page 449	global function
url_OnDisplayProcessInstance	page 450	global function
url_OnDisplayWorklist	page 450	global function
url_OnListApplications	page 451	global function
url_OnListEntryNodes	page 451	global function

Appendix B

Reserved Words

This appendix lists reserved words in Process Manager.

The following words are used internally by Process Manager and must not be used as the names of data fields. Using reserved words as field names may generate an error when an application is deployed.

Table B-1 Reserved Words in PAE

abort	cascade	datetime	for	on	start
accept	case	dba	force	open	stop
add	change	dec	form	or	table
admin	char	decimal	format	order	tables
all	character	declare	from	positive	tablespace
allocate	check	default	function	project	task
alter	checkpoint	definition	global	public	temporary
analyse	client	delay	go	quota	text
and	close	delete	goto	raise	then
any	cluster	delta	grant	range	this
append	cobol	disable	group	real	time
archive	column	dismount	groups	rename	trigger
array	columns	dispose	if	replace	true
as	comment	distinct	image	request	type
assert	commit	do	index	resume	union
assign	compile	double	indexed	reverse	unique
at	compress	drop	insert	revoke	unlimited

Table B-1 Reserved Words in PAE *(Continued)*

audit	connect	each	int	role	update
authorization	constant	else	key	roles	use
avg	constraint	enable	layer	row	user
backup	contain	end	link	schema	using
badfile	contents	entry	list	server	values
become	continue	erase	log	set	varchar
before	contraints	escape	longtext	share	varchar2
begin	count	events	manual	shared	variance
between	crash	exit	max	size	view
blob	create	explain	min	smallint	views
block	current	false	new	snapshot	when
body	cursor	file	next	some	while
boolean	cycle	fixed	not	sort	with
by	data	float	number	space	xor
cache	database	flush	off	sql	yes
cancel	date				

Glossary

activity A step in an application where an assignee needs to perform an action.

application In Process Builder, the application the builder creates to handle a process.

applet A Java component designed to run in a web browser.

assignee The person assigned to an activity for a particular process instance.

automated activity A step in an application where an action is performed automatically, without being assigned to a user.

bean A reusable software component for visual development environments.

builder The person who creates the application using Process Builder.

child process In subprocesses, the subordinate process that is called by the main or parent process.

class ID The identifier of a group of fields with some common properties.

CGI Common Gateway Interface. The specification for communication between an HTTP server and gateway programs on the server. Allows web interfaces to databases and enables the dynamic generation of HTML documents by gateway programs.

cluster The combination of a configuration directory, a corporate user directory, a relational database, a web server, and one or more iPlanet Application Servers. This combination of components is the environment for deployed applications, which are run by the application servers.

configuration directory The Directory Server where Process Manager application and cluster information is stored.

content store The place on the Enterprise Server where file attachments are stored, and the user and password needed to access them.

corporate user directory The Directory Server used to store the user and group information for a corporation. Process Manager uses it to set up users and groups and assign users work items by leveraging the directory users and groups and other attributes.

creator The person who initiates a process instance. Sometimes called the initiator.

custom activity

database The database you are using to store the information generated by process instances. For example, the database could be Informix, Oracle, or Sybase.

decision point A point at which a process map branches depending upon conditions defined in the decision point.

deploy To copy an application stored locally to a cluster. It can be deployed for storage only, or it can be deployed for development or production. Application information is deployed to the configuration directory, and the application is activated on the iPlanet Application Server.

entry point A point in the process where a user can initiate a process instance.

exception handler Used in subprocesses, a step in an application that allows the administrator to intervene manually if errors occur in the interaction between a parent and child process.

exit point A point in the process where the process ends.

extranet An extension of a company's intranet onto the Internet, to allow customers, suppliers, and remote workers access to the data.

form A part of an application a user fills out to complete a process instance, or uses to view information on a process.

group A set of users set up either in the corporate user directory or in an individual application. A group is used to assign work items to users and control which forms which users are able to see.

HTML HyperText Markup Language. A markup language (derived from SGML) used to create web documents.

HTTP HyperText Transfer Protocol. A protocol for communication between web clients and servers.

initiator The person who initiates a process instance. Sometimes called the creator.

intranet A network which provides similar services within an organization to those provided by the Internet outside it but which is not necessarily connected to the Internet.

LDIF A format for storing directory entries.

nested parallel process A parallel process nested within a larger parallel process. The activities in the nested process are considered to be part of the nested process and not the larger process.

PAE An acronym for Process Automation Edition, which was the name of the 4.0 release of this product.

parallel processing A step in an application that branches between two or more branches so that two or more activities can execute in parallel.

parent process In subprocesses, the main process that calls the subordinate or child process.

participant A user of Process Express.

Process Administrator The component of Process Manager that administrators use to administer Process Manager and Process Manager applications. Both Process Administrator, the IT administrator's interface, and Process Business Manager, the business manager's interface, are described in the *Process Administrator's and Business Manager's Guide*.

Process Automation Edition The iPlanet process management solution.

process A process is a series of steps, or work items, that can be completed by participants using an application.

Process Builder The component of Process Manager where you can design and deploy applications.

Process Business Manager The component of Process Manager that business managers use to administer Process Manager work items and process instances. Both Process Business Manager, the business manager's interface, and Process Administrator, the IT administrator's interface, are described in the *Process Administrator's and Business Manager's Guide*.

Process Engine The part of Process Manager that contains Process Express, Process Administrator, Process Business Manager and the engine that runs Process Manager. It contains all Process Manager components, with the exception of Process Builder.

Process Express The component of Process Manager that end user use to initiate process instances, complete work items, and search for process instances.

process instance A particular example of a process; for example, in a time off process application, a process instance would be a particular request by an employee for vacation time off for a specific period of time.

process map The visual representation of the process that is handled by a Process Manager application.

processing branch A set of activities that progress from a given split to its corresponding join. Also called a thread.

property An attribute of an item or component used in an application that contains information about the item. For example, an activity has properties containing information such as the name of the activity, what script is run when it is completed, and so on.

role A role is the part a user plays in a specific process instance.

script A JavaScript file. A script can include a function, but this is not a defining characteristic.

subprocess A fully functional process that is called from within another process. The process that calls the subprocess is the parent process and the subprocess is its child process.

trusted user A group that allows a secure handshake between a parent and a child process.

URL Uniform Resource Locator. The address system used by servers and clients to request web documents. It is often called a "location." The format of a URL is:

`[protocol]://[`*machine*`:`*port*`]/[`*document*`]`

For example:

```
http://docs.iplanet.com/index.html
```

transition The links between steps in a process. On the process map, they are represented by lines with arrows that lead from one item to another. Transitions can be regular or they can depend upon a condition being true before they are executed.

web publishing A feature that lets users access and manipulate server files with a server client, so that they can edit and publish documents to the web server.

work item An individual task in a process instance as it appears to the end user on a work list.

Index