

Peregrine

Open Application Architecture Platform

Tailoring Kit Guide

Version 2.2.3—For Windows

Copyright © 2002 Peregrine Systems, Inc. or its subsidiaries. All rights reserved.

Information contained in this document is proprietary to Peregrine Systems, Incorporated, and may be used or disclosed only with written permission from Peregrine Systems, Inc. This book, or any part thereof, may not be reproduced without the prior written permission of Peregrine Systems, Inc. This document refers to numerous products by their trade names. In most, if not all, cases these designations are claimed as Trademarks or Registered Trademarks by their respective companies.

Peregrine Systems® is a registered trademark and Get-Resources™ and Get-It™ are trademarks of Peregrine Systems, Inc. or its subsidiaries.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>) and by Advantys (<http://www.advantys.com>). This product also contains software developed by the following companies or individuals: Sun Microsystems, Inc., Jean-Marc Lugin, Netscape Communications Corporation, and Original Reusable Objects, Inc.

This document and the related software described in this manual are supplied under license or nondisclosure agreement and may be used or copied only in accordance with the terms of the agreement. The information in this document is subject to change without notice and does not represent a commitment on the part of Peregrine Systems, Inc. Contact Peregrine Systems, Inc., Customer Support to verify the date of the latest version of this document.

The names of companies and individuals used in the sample database and in examples in the manuals are fictitious and are intended to illustrate the use of the software. Any resemblance to actual companies or individuals, whether past or present, is purely coincidental.

If you have comments or suggestions about this documentation, please send e-mail to support-sd@peregrine.com

This edition applies to version 2.2.3 of the licensed program.

Peregrine Systems, Inc.
Worldwide Corporate Campus and Executive Briefing Center
3611 Valley Centre Drive San Diego, CA 92130
Tel 800.638.5231 or 858.481.5000
Fax 858.481.1751
www.peregrine.com



Contents

	Introducing the Peregrine OAA Tailoring Kit	9
	About this Guide	11
	Conventions Used in this Guide.	11
Chapter 1	Installing the OAA Tailoring Kit	13
	Installing the OAA Tailoring Kit	14
	Opening your Web Application Project	19
	Configuring your System for Tailoring.	19
	Setting up a Development Environment	19
	Setting Up a Testing Environment.	20
Chapter 2	Using Peregrine Studio	23
	The Studio Interface	24
	Project Explorer	25
	Drag and Drop	27
	Enabling the HTTP Listener and Form Information	28
	Viewable XML Source Code	30
	Changes Indicated with Color Text	30
Chapter 3	Studio Projects and Packages	31
	Peregrine Studio Projects.	32
	Project Components	33
	Project Component Descriptions	33
	Example of Component Hierarchy	36
	Project Files	38

Building a Project	39
XML to JSPs	39
Project Build Variables	40
Setting Build Options	41
Studio Project Packages	41
Activating and Deactivating Packages	42
Package Dependencies	43
Setting Package Dependencies	43
Saving Changes with Package Extensions	44
Warnings for Conflicts	45
Deploying to UNIX Platforms	46
Requirements	47
Configuring for FTP Deployment	47
Deploying via FTP	47
International Builds	48
Configuring for an International Build	48
Exporting Strings for Translation	49
Importing Strings for Translation	51
Adding to an Existing Frameset	52
Chapter 4	
Forms and Form Components	53
Tailoring Forms	54
Changing Form Titles	55
Changing Form Instructions	56
Changing a Form's Onload Script	57
Changing Form Component Labels	59
Hiding Form Components	60
Changing a Form Component to Read-only	61
Changing the Schema that a Form Component Uses	62
Changing the Document Field that a Form Component Uses	63
Specifying a Document Field Name	64
Displaying Forms within a Frameset	67
Types of Form Components	69
Component Template Containers	69
Fieldsection Containers	70
Text Edit Fields	71

	Selectbox Fields	72
	Hidden Data Fields	74
	Redirections	74
	Table Form Components	75
	Table Links	76
	Text Columns	77
	Actions	77
Chapter 5	Adding Personalization Functionality	79
	Supporting Personalization	80
	Activating Personalization	80
	Personalization Hierarchies.	82
	Personalizing with DocExplorers	82
	DocExplorer Forms and Functions	83
	Adding a DocExplorer Reference	83
	Personalizing DocExplorer	84
	Adding Personalization to Lookup Fields	85
	Creating a Nested Document Lookup	87
	Using the Personalization Interface	88
	Adding Fields to a Form	89
	Configuring Field Attributes	89
	Removing Fields from a Form	90
Chapter 6	Scripting	91
	Types of Scripts.	92
	Where Scripts are Stored.	92
	How Scripts are Used	93
	Where Scripts are Used	94
	Testing Scripts with URL Queries	95
	URL Script Queries Template.	96
	URL Schema Queries Template	96
	Using Variables to Provide Script Data.	97
	Common Message Operations	100
	About Script Pollers	102
	Enabling Script Polling	102
	Stopping Script Polling	104

	Script Pollers in a Multiple JVM Environment	104
	Sample Scripts	105
	General Script Samples	105
	Selecting a Field from a Schema	105
	Calling Other Scripts and Combining the Results	107
	Form Script Sample	109
	Creating an XML Document from a Schema	109
	Script Poller Sample	112
	Maintaining a Connection to AssetCenter	112
	References	115
	Sources for Client-side JavaScript	115
	JavaDocs for the Main Archway Package	115
Chapter 7	Document Schema Definitions	117
	How Schemas are Used.	119
	Schemas with ECMAScript	119
	ECMAScript Syntax	120
	Identifying the Back-end System Version	121
	AssetCenter Feature Links	122
	Schema Naming Conventions.	123
	Schema Elements And Attributes	123
	<schema>	123
	<documents>	123
	<document>.	125
	<document> attributes	125
	name	125
	table	125
	field	126
	joinfield	126
	joinvalue	126
	<attribute>	127
	name	127
	type.	127
	Type values	128
	shortdesc	129
	search.	129

	list	129
	detail	129
	create	129
	field	130
	link	130
	linktable	130
	linkfield	130
	linktype	131
	linkkey	131
	How to Create Schemas	132
	Creating Groups of Schemas	132
	Creating Schemas	132
	Schema Template	132
	Schema Template Entry Descriptions	133
	Using Nested <Document> Elements to Call Linked Tables	134
	Nesting the <Document> Element In-place	135
	Nesting the <Document> Element by Reference to <Collection>	136
	Nesting the <Document> Element by Reference Docname	139
	Frequently Asked Questions	141
	How do I create document queries to linked tables?	141
	Why do all query response messages contain ID elements?	143
	Can schemas include SQL statements?	143
	How are schema definitions converted into SQL statements?	143
Appendix A	Peregrine Studio Components	149
Appendix B	Troubleshooting and FAQs	161
	Web Application Environment	161
	Out of memory error	161
	Peregrine Studio	162
	Cannot edit — components are displayed with grey background	162
	Red exclamation point (conflict icon) displayed next to nodes	163
	Import Errors	165
	Unable to import customized files	165
	Bad magic number	165
	Scripting Errors	166

Cannot find script file	166
Script produces an ECMAScript error	167
ECMAScript error: undefined value or property	167
Web Application Errors	168
Wrong start form is displayed for activity.	168
Script output not appearing in form component.	168
Too few parameters error	169
Web application always goes to redirection form.	170
JDBCCalls error at login	170
Syntax error in FROM clause	170
Index	173

Introducing the Peregrine OAA Tailoring Kit

The OAA Tailoring Kit includes:

- Peregrine Studio
- Source files for your Web applications

The OAA Tailoring Kit is intended for Web application developers who are familiar with Extensible Markup Language (XML), ECMAScript, Structured Query Language (SQL), and back-end database systems such as AssetCenter and ServiceCenter.

Peregrine Studio is a graphical development tool that you can use to customize Web applications built on the Peregrine OAA Platform. Peregrine OAA Platform Web applications are a series of Web-based interfaces that allow users to, for example, order and purchase goods, search for documents, and submit requests. The Peregrine Portal common login determines what portions of the interface are dynamically generated for a user.

The Web-based interfaces are the result of the following components:

- A collection of Java Server Pages (JSPs) that provide the browser interfaces for the Web application. The Web application JSP content is created during the Studio build process.
- A Web server to host the Web application JSP content.
- A Java-enabled application server to run the Archway servlet. The Archway servlet routes and formats data requests between Web applications and back-end database systems.
- A collection of ECMAScripts that allow for dynamic parsing and formatting of Web application data sent to and received from the client Web browser.

From an administrative perspective, Peregrine OAA Platform Web applications are the output of one or more Studio project files. Studio elements such as packages, modules, activities, and forms describe the end-user interface. Other Studio elements such as ECMAScripts and document schema definitions determine what data the Web application interfaces receive or process from back-end databases.

The Web application produced during a build is the result of the following Studio components:

- A project file that describes all the Web applications available. Each project file contains its own list of Web applications that you can use to produce and deploy an installation.
- Components that define the functionality of each Web application. All Web applications are built from packages, modules, activities, forms, and form components. Each of these components is saved as an XML file in the project.
- A back-end database or application to store the data accessed by Web application forms, track workflow tasks, and store personalization changes. Typical back-end systems include AssetCenter, ServiceCenter, and JDBC-compliant databases.
- Document schema definitions used by the Archway servlet to format message objects sent to and received from back-end databases. All Archway message objects are formatted as XML documents.
- ECMAScripts to generate and send message objects to the Archway servlet. The messenger objects can be used to query back-end databases for specific data and format the results for display or processing by Web application forms.

About this Guide

This guide is intended for use by a developer who will be tailoring a Web application built on the Peregrine OAA Platform.

This guide should be used in conjunction with several other manuals, which are:

- The guides for the Peregrine Web applications you have installed.
- Documentation for the Peregrine back-end systems you are using.
- Documentation for the application server you are using.

Conventions Used in this Guide

Screen shots in this guide are included as examples only. A sample Web application, Employee Lookup, is used for the Studio screens. Get-Resources forms are shown using the Classic theme.

The following documentation conventions are used in this guide:

Object	Example
Button	Click Next
File name	The <code>login.jsp</code> file
Sample script or XML code	<pre>var msgTicket = new Message("Problem");</pre> <p>...</p> <pre>msgTicket.set("_event", "epmc");</pre> <p>The ellipsis (...) is used to indicate that portions of a script have been omitted because they are not needed for the current topic. Samples of code are not entire files, but they are representative of the information being discussed in a particular section.</p>
Menu option	Select Start>Program Files.
Book title	Refer to the <i>Get-Resources Installation and Administration Guide</i> .

1 Installing the OAA Tailoring Kit

CHAPTER

The OAA Tailoring Kit installation will install Peregrine Studio and the source files for your Web application.

Before you begin the installation, your Web application and associated software (application server, back-end system) should already be installed.

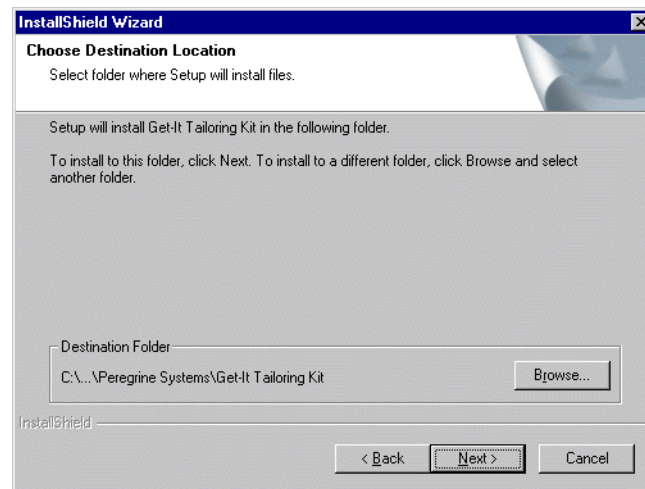
Installing the OAA Tailoring Kit

The installation process has two sections:

- In the first section, the source files for your Web application are copied to the location you specify.
- In the second section, Peregrine Studio is installed.

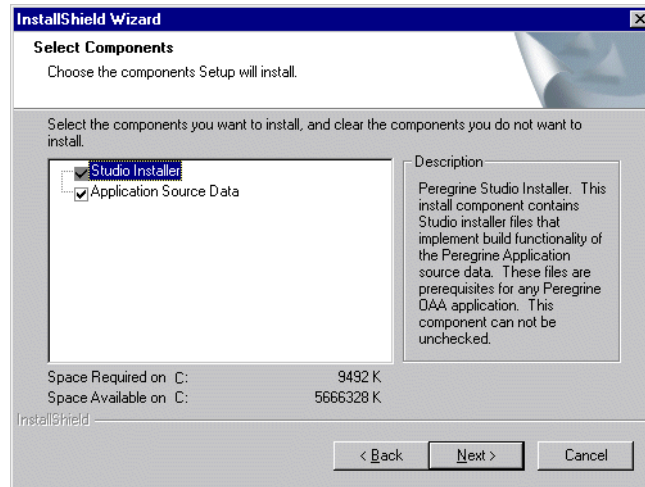
To install the OAA Tailoring Kit:

- 1 Insert the installation CD into the CD-ROM drive.
Setup is launched, and the Welcome dialog box is displayed.
- 2 Click **Next**.
- 3 In the License agreement dialog box, click **Yes** to accept the terms.
The Destination Location dialog box is displayed.

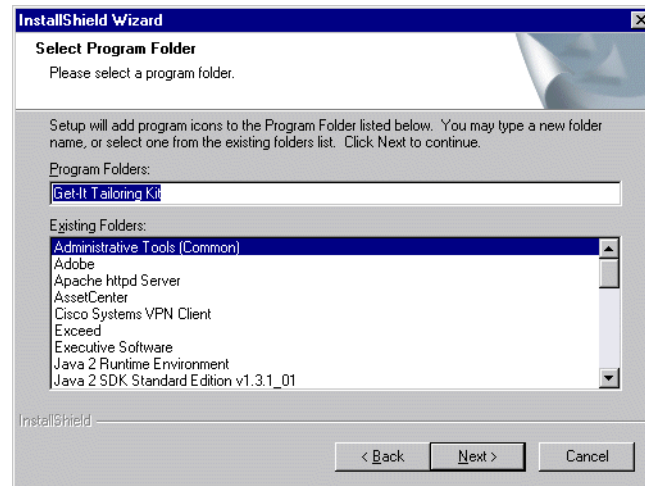


- 4 Click **Next** to install the source files to the default location, or click **Browse** to select another location, and then click **Next**.

The Select Components dialog box is displayed.

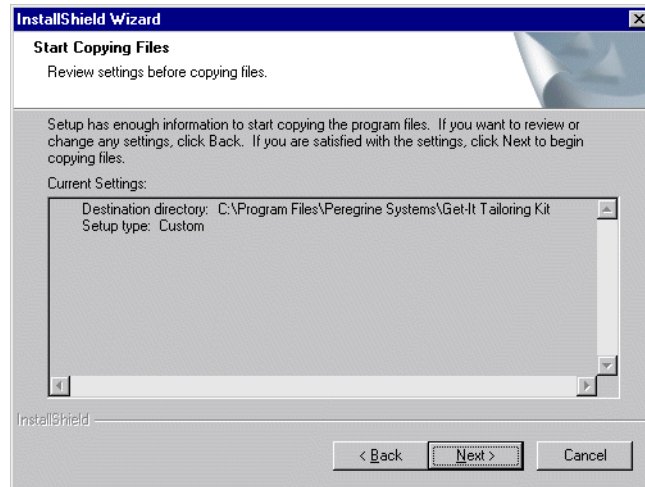


- 5 Verify that both components are selected, and then click Next. The Select Program Folder dialog box is displayed.



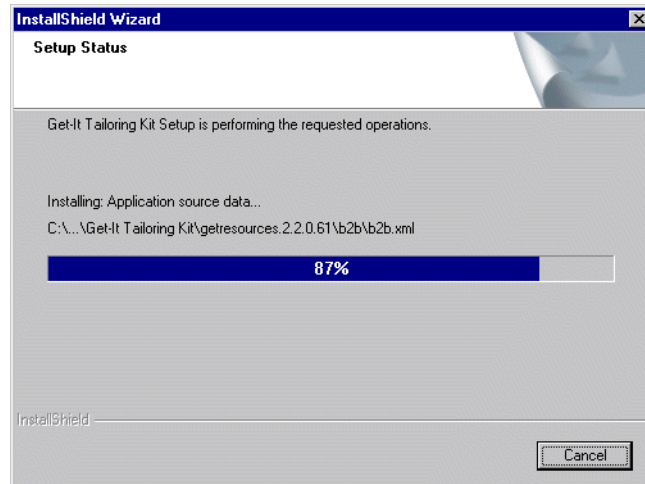
- 6 Click Next.

The Start Copying Files dialog box is displayed.



- 7 Verify that the information is correct, and then click **Next**.

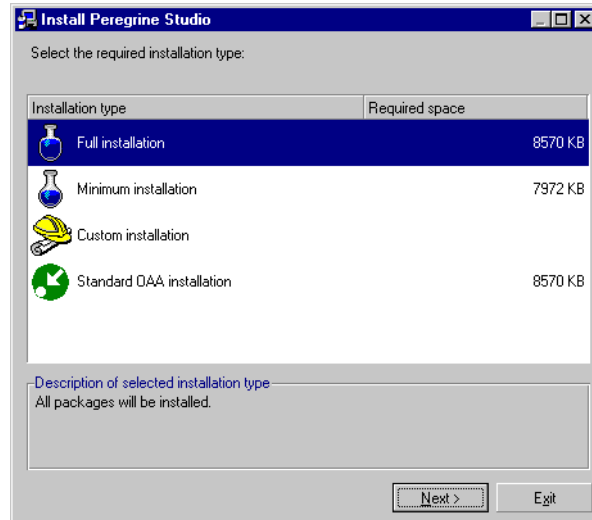
A Setup Status dialog box is displayed, indicating that the files are being copied.



When the files have been copied, the installer for Peregrine Studio is launched.

- 8 The Welcome dialog box recommends closing all active applications. When you have done this, click **Continue**.

- 9 In the Installation Type dialog box, ensure that Full installation is selected (the default), and then click Next.

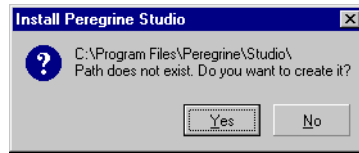


A summary dialog box is displayed.



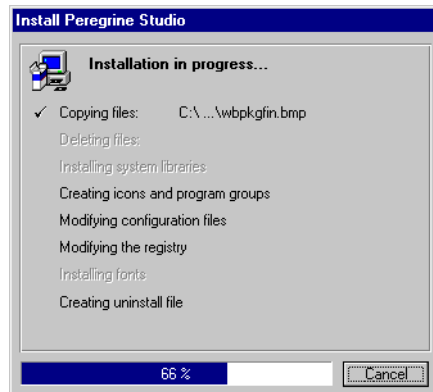
- 10 Verify that the information is correct. If necessary, browse to another location where you want the files installed. Click Install.

A prompt is displayed that the path you have chosen does not exist.

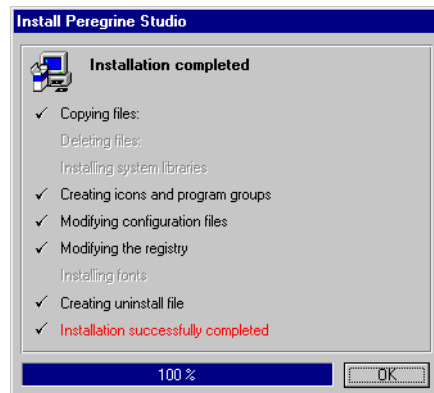


- 11 Click Yes to create this folder.

A dialog box is displayed indicating that files are being copied.



- 12 When a message is displayed that the process is complete, click OK.



- 13 Click Finish in the InstallShield Wizard.

Opening your Web Application Project

After the installation is complete, you can open your Web application project in Peregrine Studio using the following procedure.

Important: If you have not already received a Studio authorization file, contact Peregrine Customer Support. You will need this file in order to edit your Web application files.

To open your Web application source files in Studio:

- 1 Go to Start>Programs>Peregrine>Studio>Peregrine Studio to open Studio.
- 2 From the Tools menu, select **Authorization file**.
- 3 In any text editor, open the authorization file provided for Studio.
- 4 Copy the contents of the authorization file into the Authorization file dialog box in Studio. Click OK.
- 5 From the File menu, select **Open project**.
- 6 Browse to the location of the .adw file for your Web application. For example, the default location for the `get-resources.adw` file is:
C:\Program Files\Peregrine Systems\GetItTailoringKit\getresources 2.2.0.61
- 7 Click **Open**.

Configuring your System for Tailoring

You can set up one or more development environments separately from your deployment platform. A development environment lets you modify Web applications from one computer system and build the Studio project to a separate test or deployment environment.

Setting up a Development Environment

You need the following minimum components for a development environment:

- Peregrine Studio.
- Java Runtime Environment 1.3 or later (necessary to run Studio), or the Java Development Kit provided with your Web application installation.

- Studio project files.
- Java Development Kit 1.2.2 or later if you want to create or edit your own wizards for Studio.

With this minimal development environment, you can modify Web applications using the built-in Studio tools and wizards. You can then do one of the following:

- Build your Studio projects on the development computer and copy the results to a deployment computer
- or
- Enter the network path to the deployment computer in your Studio Build Settings.

Important: If you are using source control software to store your project files, you will need to configure your Studio Project Settings to check out and check in the source files.

Setting Up a Testing Environment

You need the following components to test or debug:

- Peregrine Studio.
- Java Runtime Environment 1.3 or later (necessary to run Studio).
- Your Web application project files.
- If you want to create or edit your own wizards for Studio, you will need to install a Java Development Kit 1.2.2 or later. The Java 2 SDK Standard Edition v1.3.1_01 is provided on the installation CD for your Web application.
- Your Peregrine Web application.
- Web server (necessary to serve Web application JSP content).
- Java-enabled application server (necessary to run the Archway servlet). Tomcat is provided on the installation CD for your Web application.
- JavaScript-enabled Web browser (necessary to view your Web application).

With this testing environment, you can build and view your Web application changes from a single computer. To set up a testing environment, perform a full installation. Refer to the installation guide for your Peregrine Web application for instructions and supported software versions.

You can test several configurations by storing each configuration as a separate package extension within a project file. You can then activate a package extension and build the project as you are ready to test each configuration.

2 Using Peregrine Studio

CHAPTER

This chapter provides an overview of the Studio interface. The primary focus is on those elements most often used for Web application development and debugging. For more information about configuring or using Studio, refer to the Studio online help.

This chapter covers the following topics:

- An introduction to the Studio interface, and how to use the Project Explorer to navigate through a Studio project file and use context-sensitive menus to change individual components.
- How to enable Form Information to easily access the form you want to modify in Studio.
- How to view the source code in Studio.

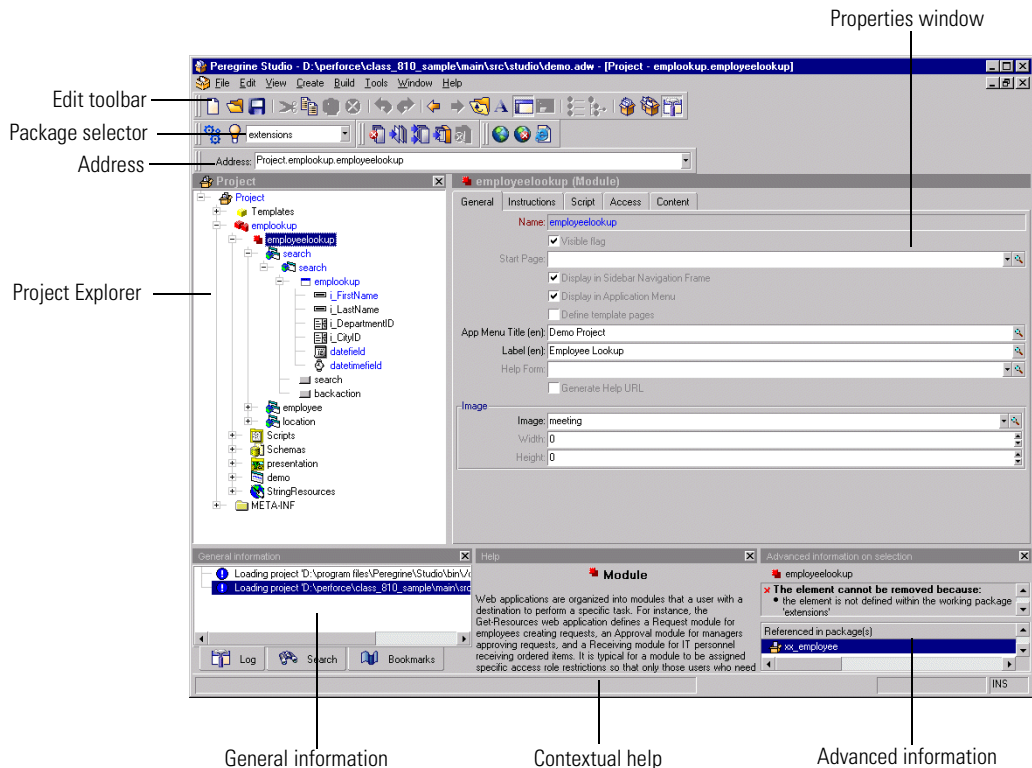
Note: A sample Web application, Employee Lookup, was created to illustrate how Studio is used. Sample schemas and scripts are provided throughout the guide using the Employee Lookup application as an example.

The Studio Interface

The Studio interface includes:

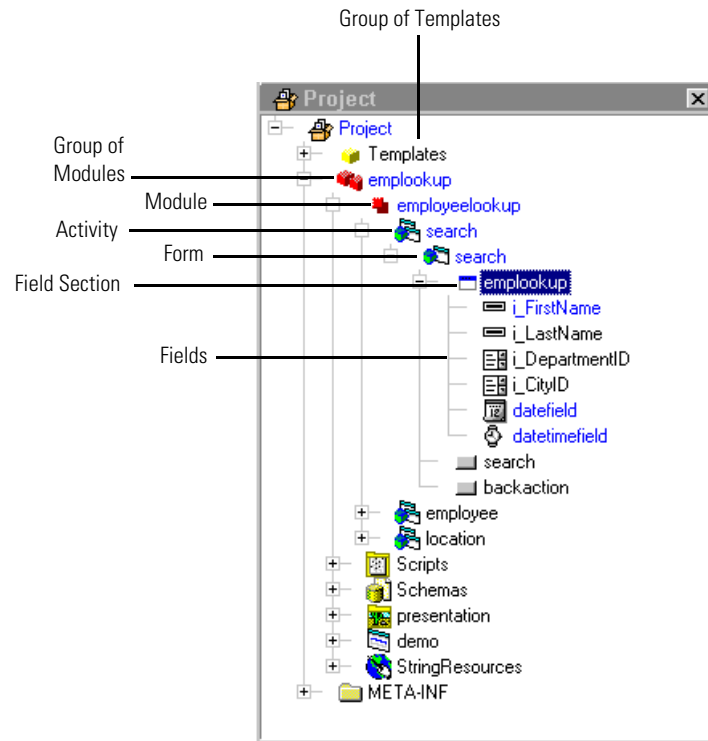
- Project Explorer
- Properties window
- Edit toolbar
- General information display
- Contextual help
- Address
- Package selector
- Advanced information

All elements of the interface except the Project Explorer and the Properties Window can be hidden by unselecting them on the View menu.



Project Explorer

The Project Explorer provides a hierarchal view of all the components that comprise a Studio project. The Project Explorer window displays each component as a separate node within the tree.



Left-click a node

Click the node listing the component you want to change and the properties of the component display in a window of the Properties pane.

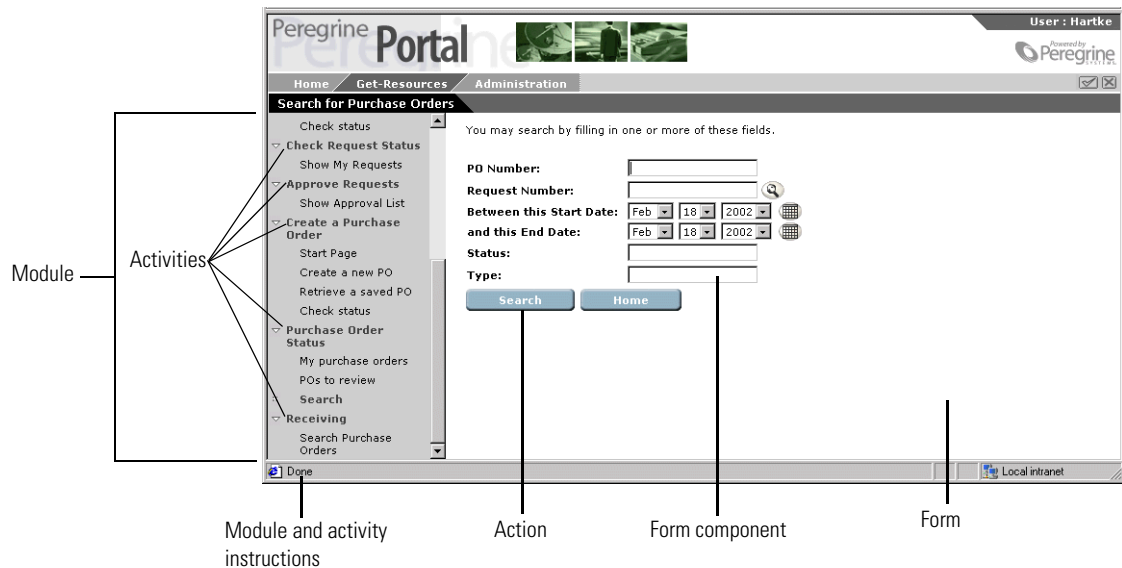
Right-click a node

Right-click a node to display a list of context-sensitive options.

The options listed in the following table are available for all nodes.

Menu item	Description
New (and/or Open)	Provides a context-sensitive menu of allowed components that you can add from the current node. The list of components in this menu is dynamically updated for each node of the Project Explorer tree.
Open	Displays the properties of the selected component in a window of the Properties pane.
Open in New Window	Displays the properties of the selected component in a new window of the Properties pane.
Rename	Renames the selected node to the new name typed by the user. This option will only be available when a package extension has been activated as the save location for changes.
Cut	Removes the selected node, and all child nodes underneath, and places a copy in the Windows clipboard.
Copy	Copies the selected node, and all child nodes underneath, to the Windows clipboard.
Paste	Inserts the contents of the Windows clipboard. If the clipboard contains a Studio component, it will be automatically placed within the tree according to the type of component it is.
Delete	Deletes the selected node and all child nodes. This option will only be available when a package extension has been activated as the save location for changes.
Help	Displays the Studio help system.
Export node	Saves a copy of the selected node, and all child nodes underneath, as an XML file, which can be imported into a Studio project.
Import node	Opens a user-selected XML file describing Studio nodes and inserts it into the tree. The imported node will be inserted below the node you right-clicked.
Add Bookmark	Adds a bookmark link to the node you currently have open in Studio. If you browse to another location and then want to return to this node, click the Bookmarks tab in the General Information window and select the appropriate bookmark.

The following image shows how some of the common Studio components are displayed in a Peregrine Web application interface.



The address bar

You can use the Address Bar to navigate directly to any Studio project component. The address bar will display as a text box below the Edit Toolbar.

To display the address bar:

- 1 Open Studio.
- 2 Click View > Address Bar. The Address Bar displays below the menus.

Drag and Drop

Studio supports drag and drop movement of components within the Project Explorer. Changing the order of nodes in the Project Explorer will change how the items are presented in the Studio build.

To move a component within the Project Explorer:

- 1 Click and hold the left mouse button over the name of the node you want to move.
- 2 Drag the node to the new location in the Project Explorer tree.

The node appears underneath the component (of the same level) where you drop the node.

Note: You cannot move components out of the order enforced by the DSD. For example, you cannot move a form out of an activity and place it at the same level as a module. You can, however, change the order of the forms listed under an activity.

Enabling the HTTP Listener and Form Information

Using the HTTP Listener, you can click on the Form Information address listed for a given form and the appropriate form properties will be displayed in Studio. This debugging feature allows you to navigate through Web applications with a browser and quickly bring up any particular form that needs modification.

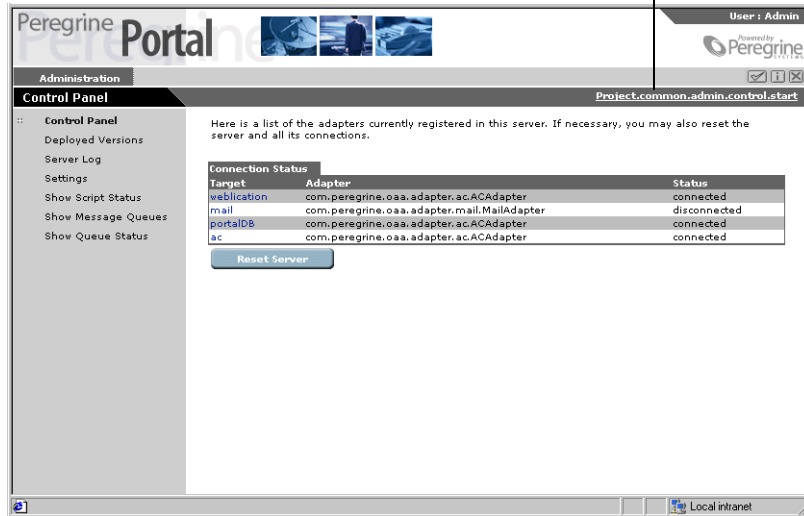
To enable the HTTP Listener and Form Information functionality:

- 1 Enable the HTTP listener as follows:
 - a Open Studio.
 - b Click Tools > Options.
 - c Select the **Use listener** check box from the HTTP Listener section.
 - d Select the port number you want the HTTP listener to use (the default port is 81), and then click **OK**.
 - e Save your Studio project.
- 2 Open the project file containing the form you want to change in Studio.

Note: Be sure to select or create a package extension in which to save any changes.
- 3 Log in to your Web application as an administrator, or access the Admin module directly from the Administrator login page (`admin.jsp`).
- 4 Click Admin > Settings to display the Settings form.
- 5 On the Common tab, set the **Show form info** setting to *true*.
- 6 Click **Save** at the bottom of the form to activate your new settings.

- 7 On the Control Panel form of the Admin module, click **Reset Server** to commit your changes.


With Show Form Info enabled, click this link to open the form in Studio.



- 8 In your Web application, navigate to the form you want to tailor.
- 9 Click the Studio address displayed in the Form Information banner of the Web application form.

Studio will appear as the active window and display the current form's properties page.

Viewing Referenced Components

Whenever an item links to or references another component, Studio will display a magnifying glass button  next to the field.


You can click this button to display the form, image, schema, or script that is called by the reference.

Use **Go to Previous View**  (the orange arrow pointing to the left) to return to the component making the reference.

Viewable XML Source Code

All Web application information is saved in XML files that you can see from Studio. Studio does not support direct editing to the XML source of Web application components. All XML source views are listed with a grey background which indicates that the item is read-only.

To view the XML source code of a Web application:

- 1 Select the node of the Web application or component you want to view from the Project Explorer.
- 2 Click the **Source view** button  (the blue capital A).
The XML source appears in the Properties window. The XML source code is color coded as you define in the project settings.

Changes Indicated with Color Text

All changes or additions you make to your Studio project are highlighted with blue text.

Within the Project Explorer view, Studio highlights each node of the tree that contains a component that has been changed or added. This allows you to navigate through the Project Explorer tree view and locate where you have made changes and additions.

To view the changes made in a project:

- 1 Select a node displayed with blue text to view the component properties.
- 2 Review the properties listed in the window displayed to the right of the Project Explorer (the Properties window). Changes that were made to this component will be displayed with blue text. If no blue text is displayed in the Properties window, then the change or addition is in one of the child nodes below the current node.
- 3 If necessary, expand any child nodes highlighted with blue text and review the Properties window for changes.

3 Studio Projects and Packages

CHAPTER

Peregrine Studio *projects* contain all of the *packages* that make up an application. A new package, or multiple packages, must be created when you are making changes to your project. These packages can then be activated or deactivated, depending on which features you want to be included in your current project.

Packages are not displayed in the Project Explorer *Project* tree. The list of available packages (packages that have been activated) is included in the Package Explorer drop-down list located below the toolbar in Studio.

Peregrine Studio Projects

Studio saves all the source files for your Web applications as a project. A Studio project consists of several components that are combined during the build process.

Studio component	Description
Web application components	The XML files that define the functionality of your Web applications. All Web applications consist of packages, modules, activities, forms, and form components.
ECMAScripts*	ECMAScripts create and format message objects to the Archway servlet. Web application components will use ECMA message objects to display and process data.
Document schema definitions	The XML files that define how the Archway servlet should format the ECMA message objects sent to and received from back-end databases. Web application components will use the ECMA message objects to display and process data.
Presentation files	Any supporting files such as images, client-side JavaScript, hand-coded HTML or JSP files, or translation strings that will be included with the Web application.
Stylesheets	The Cascading Style Sheet (CSS) files that define the colors and fonts that will be used in your Web application pages.

*ECMAScript is the core language standard shared between the JavaScript and JScript libraries.

Project Components

Studio organizes Web application components into a hierarchy of parent and child elements. The hierarchy of Web application components defines the individual properties of each Web application component.

Properties include, for example, the valid parent-child configurations of a component and the type of editor each component requires within Studio. All Studio projects conform to the same hierarchy.

Project

Packages - Group of modules (saved as system packages and user packages)

Supporting files (templates, scripts, schemas, images, strings, etc.)

Module







Activity (or DocExplorer)




Form

Form components (action, field, table, etc.)

Project Component Descriptions

This table lists and describes some of the Studio components. For a complete list of the components that make up a Studio project, refer to Appendix A, beginning on page 149.












Component	Description
Project	<p>The project component:</p> <ul style="list-style-type: none"> ■ is the container for all the elements that are part of your current project file. ■ is always the top node of the Project Explorer tree. ■ is represented by an open package icon () in the Project Explorer tree.
Templates (support files)	<p>The templates component:</p> <ul style="list-style-type: none"> ■ is the container for all the Web application elements that can be reused throughout the project. ■ appears with a yellow cube icon () in the Project Explorer tree.
Group of modules	<p>The group of modules component:</p> <ul style="list-style-type: none"> ■ is the container for all the supporting files and modules that make up a Web application. ■ appears with a double red cubes icon () in the Project Explorer tree. ■ does not have any one dedicated graphical representation in the built Web application.
Module	<p>The module component:</p> <ul style="list-style-type: none"> ■ is a container for the activities and forms that make up a Web application. ■ appears with a double red box icon () in the Project Explorer tree. ■ appears as a text link on the navigation sidebar and may also appear on the Web application Menu. <p>The module component is usually where access restrictions are defined. Setting access restrictions limits a module to particular user roles.</p>
Activity	<p>The activity component:</p> <ul style="list-style-type: none"> ■ defines a particular Web application task or action such as searching for records, displaying records, or entering records. ■ is a container for a particular set of Web application forms. ■ appears with a cube and two window panes icon () in the Project Explorer tree. ■ appears as a text link on the navigation sidebar (Activity Menu).
Form	<p>The form component:</p> <ul style="list-style-type: none"> ■ is where Web application user interfaces and displays are defined. ■ appears with a cube and a single window pane icon () in the Project Explorer tree. <p>Typically, the system displays each form component as a page in the main Web application frame.</p>








Component	Description
Form components	<p>Form components such as fields, actions, tables, and lookups define the actual user interfaces and displays used in a Web application form.</p> <p>Form components appear with a variety of icons in the Project Explorer Tree.</p> <p>Most form components also have a graphical element in the Web application form.</p>
Group of scripts	<p>The group of scripts component:</p> <ul style="list-style-type: none"> ■ is a container for all the server-side ECMAScripts used by a Web application. ■ appears with a document with a yellow border icon () in the Studio Project Explorer tree.
Group of schemas	<p>The group of schemas component:</p> <ul style="list-style-type: none"> ■ is a container for all the document schema definitions that a Web application uses. ■ appears with a data store and document icon () in the Studio Project Explorer tree.
Group of files	<p>The group of files component:</p> <ul style="list-style-type: none"> ■ is a container for supplemental files that your Web applications can use. You can store images, client-side JavaScript, localized string files, or initialization files here. ■ appears with a folder icon () in the Studio Project Explorer tree.

*Portal Components are available only in the portal module.

Example of Component Hierarchy

This table shows how the hierarchy could be applied, using a sample travel Web application as an example.








Component	Description
Group of modules  travel	This is the top node of the Travel Web application.
Module  reservations	This module contains all the activities necessary to plan, reserve, and review the status of travel reservations. This module is accessible to all users.
Activity  reservation	This activity is the container for all travel reservation tasks.
Form  search	This form allows users to search for and book flights and ground transportation by date and carrier.
Form  billing	This form allows users to enter billing information.
Activity  confirmation	This activity is the container for all review and confirmation tasks.
Form  confirm	This form displays all the travel reservations a user has made and allows each trip to be confirmed, modified, or cancelled.
Activity  status	This activity is the container for all reservation status requests.
Form  current	This form allows users to access the current status of a reservation request and where it is in the approval process.
Module  approvals	This module contains all the activities necessary to review and approve travel reservations. This module is limited to users with an approver access right.
Activity  review	This activity is the container for all search and review tasks.

Component	Description
Form  search	This form allows approvers to search through a list of travel reservations needing approval.
Form  results	This form displays a list of travel reservations that meet a given search criteria.
Form  details	This form displays detailed information about the travel reservation such as the person who requested it, the travel times, and the billing information entered.
Activity  confirm	This activity is the container for all confirmation activities.
Form  approve	This form allows approvers to approve travel reservations.
Form  update	This form allows approvers to change travel reservations before approving them.
Form  reject	This form allows approvers to reject a travel reservation and enter reasons that the reservation was rejected.

Project Files

This table describes the files that make up a Studio project and the information they contain. Items listed in italics are variables. To determine the actual file name, replace the italic text with the component name.

Warning: Do not edit these files outside of Studio. Manual changes you make outside of Studio may be lost during the build process.

Component	Saved as	Contains
project 	..\Peregrine Systems\GetItTailoringKit\ <i>project</i> \ <i>project</i> .adw	<ul style="list-style-type: none"> ■ <package> names ■ Path to package.xml
package 	Base system packages <ul style="list-style-type: none"> ■ ..\Peregrine Systems\GetItTailoringKit\ <i>web application name</i>\package\package.xml User package extensions <ul style="list-style-type: none"> ■ ..\packages\getit\package\ package.xml 	<ul style="list-style-type: none"> ■ <package> name ■ <modules> name ■ <module> names ■ Path to module.xml ■ Schema Names ■ Path to schema.xml ■ Script Names ■ Path to script.xml
modules 	part of ..\Peregrine Systems\Get-ItTailoringKit\ <i>package</i> \package.xml	<modules> name
module 	..\Peregrine Systems\Get-ItTailoringKit\ <i>package</i> \modules\ <i>module</i> .xml	<ul style="list-style-type: none"> ■ <module> name ■ XML code for <activity>, <form>, and <form> components
schema 	..\Peregrine Systems\Get-ItTailoringKit\ <i>package</i> \modules\Schemas\schema.xml	XML code for <schema>
script 	..\Peregrine Systems\Get-ItTailoringKit\ <i>package</i> \modules\ServerScripts\script.xml	XML code for <script>
presentation files 	..\Peregrine Systems\Get-ItTailoringKit\ <i>package</i> \modules\Presentation	Directory where presentation files can be stored to be included in a Studio build.

Building a Project

During the build process, Studio compiles the XML components of your project into a collection of Java Server Pages (JSP). The JSP content will be generated in whatever folder you selected in your Studio Build Settings. Generally, Studio produces one JSP for each form in the Web application.

XML to JSPs

This table summarizes how the XML content of your project is converted into JSPs during the build process. Items listed in italics are variables. To determine the actual file name, replace the italic text with the component name.

Warning: Do not edit these files outside of Studio. Manual changes you make outside of Studio may be lost during the build process.

Component	Built as
package	A collection of JSP files as detailed by the components listed below.
modules	part of <i>..\webapps\oaa\apphead.jsp</i>
module	part of <i>..\webapps\oaa\apphead.jsp</i>
activity	part of <i>..\webapps\oaa\apphead.jsp</i>
form	<i>e_module_activity_form.jsp</i>
form components, fields, redirects, tables, etc.	as part of <i>e_module_activity_form.jsp</i>

Example

Modules name = *search*

Module name = *employees*

Activity name = *empsearch*

Form name = *search*

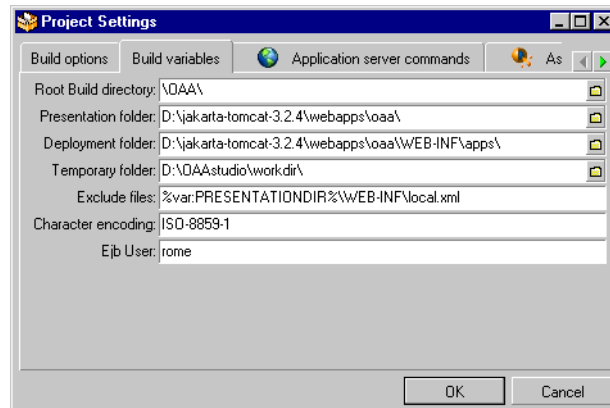
XML files saved in project = *search.xml* and *employees.xml*.

JSP created during build = *e_employees_empsearch_search.jsp*.

Project Build Variables

Studio uses the following variables to build a Web application:

- Root Build Directory—the root folder for the build.
- Presentation folder—the directory from which your Web application Java Server Pages (JSP) will be generated and run. You will need to create a Web server virtual directory mapping to this directory to run your Web application.
- Deployment folder—the directory from which your ECMAScripts and schemas will be generated and run. Normally, this is the WEB-INF\apps directory under the Presentation folder.
- Temporary folder—the directory where Peregrine Studio will generate temporary files used in the build process. You can use the variable `workdir` to save temporary files to your Windows NT profile temporary directory.
- Exclude files—a semicolon-separated list of files or directories you want to be excluded during the cleaning process.
- Character encoding—Not used. JSP encoding is determined by the character encoding setting on the Settings page of the Admin module.
- Ejb User—Enterprise Java Beans. The BizDoc database user (rome).



Setting Build Options

To set build options:

- 1 From the Build menu, select Project settings.
- 2 Click the Build variables tab.
- 3 Enter or browse to the proper directory for each of the three settings.
- 4 Click OK to save your settings.

Studio Project Packages

Packages are the Web application components that are installed as part of your Web application suite. Packages contain all the XML documents, ECMAScripts, and schemas necessary to run your Web applications. To change or add to your Web applications, you create package extensions, which are XML documents that list the changes or additions you make to your existing Web applications.

Each Web application project is defined by one or more packages. Packages are either *system* or *extension*.

- System packages. The system packages provided by Peregrine define the base functionality of a Web application.
- Extension packages. Packages you create are called *extensions*. Package extensions store all of your additions or modifications to a Web application. Extensions store only the elements that you have added or changed from the base system package.

You can see the base packages and the extensions that make up your project from the Package Activation toolbar. This view displays the active packages that can be edited and built in your project. When a package is activated, the changes or additions will be included in the build. When a package is deactivated, the changes or additions will not be included in the build. The modular design of packages allows you to decide which changes, additions, or entire Web applications will be included or excluded from the build process.

Tip: Group similar Web application functions in the same package extension. This will allow you to activate or deactivate groups of functions using the Package Activation toolbar. For example, if you are testing different interfaces with the same functionality, you may want to save each interface in a different package extension. After you determine which interface is better, you can implement the new interface by deactivating the other test package extension and rebuilding the project.

Activating and Deactivating Packages


You can control the packages and package extensions that are part of your Web application suite by activating or deactivating them from the Package Activation menu. To include a package in your Web application installation, activate the package, and then build the Studio project. To remove a package from your installation, deactivate the package and then rebuild the Studio project.

To activate a package:

- 1 To display the package activation toolbar, click View, and then click Package Selector.


The Package Activation toolbar is displayed.



- 2 Click the **Package activation** button ().
- 3 Select the checkbox next to the package name or names you want to activate.
- 4 Click OK.

All active packages will be included in the next build.

To deactivate a package:

- 1 Click the **Package activation** button ().
- 2 Clear the check box next to the package name or names you want to deactivate.
- 3 Click OK.

All deactivated packages will be excluded from the next build.

Package Dependencies

Each package has a list of dependencies that define what other packages are affected by a given current package's changes or additions.

When you create package extensions, you must select the other packages that your new extension will be dependent on. You will be able to make changes or additions to only the packages that are listed in your extension's dependencies. If you try to make changes outside your extension's dependencies, you will produce a conflict.

You can use the package dependency list to determine what other packages a particular extension affects. This information is particularly useful if you are trying to resolve conflicts in your projects.

Package dependencies are first defined by the New Package wizard when you create a package. You can manually change the package dependencies using the procedures described below.

Setting Package Dependencies

To set package dependencies:

- 1 Go to Tools > Package Dependencies.
- 2 From the left pane, select the package name for which you want to set dependencies.

The list of defined dependencies appears in the right pane.

- 3 Select the check boxes next to the package names you want to add as package dependencies. Clear the check boxes next to the package names you want to remove as package dependencies.

Note: Dependent packages activate or deactivate as a group. For example, suppose you create a user extension called `New_Interface` that is dependent on the `Extension` package. If you deactivate the `Extension` package, you will also deactivate the `New_Interface` package. If you activate the `New_Interface` package, you will also activate the `Extension` package.

- 4 Click **OK** to set the dependencies.

Saving Changes with Package Extensions

All additions and changes to a project must be saved under a package extension name. By default, all of the packages that ship with the Peregrine Web applications are write-protected and cannot be used as the save location for your tailoring efforts. To tailor your installation you need to create one or more new package extensions where your changes and additions will be saved.

Tip: By default, all new projects automatically include an Extension package extension to store your changes. However, you may want to create additional package extensions to group all changes of a particular Web application into one package extension. For example, you could create one package extension to store all Get-Resources changes and another package extension to store all Get-Services changes.


To create a new package extension:

- 1 Open Studio.
- 2 Click File > New package to start the Create New Package wizard.
- 3 Enter the name and package dependencies for the new package.
 - **Name.** Enter a name for the new package extension. The package extension name cannot contain spaces or special characters.
 - **Dependencies.** Select the existing package or packages that your package extension will be dependent on. Your new package extension must be dependent on at least one existing package. Clear the check boxes of the packages that you do not want your new package to be dependent on.
- 4 Click OK to complete the wizard.

Tip: If you are creating new Web applications, select the shared templates package as your package dependency.

The new package extension is automatically activated and selected from the Package Selection menu. Any changes or additions you make to your Web application will now be saved in your new package.

Warnings for Conflicts

Studio validates your project and ensures that there are no conflicting instructions or missing components. If Studio encounters a conflict, it displays an exclamation point  icon next to each node that contains a conflicting component within the Project Explorer view.

Studio will display a conflict warning if any of the following conditions occur.

- Two or more active components describe the same thing. For example, if you have two active package extensions that rename the same button or interface component, you will create a conflict.
- You make changes or additions to a package that is not defined as a dependent package. For example, if you create a package called *test* that is solely dependent on the package *common*, then the test package cannot make changes or additions to other packages, such as resources. Attempting to make such changes will create a dependency conflict.

Resource conflicts

Resource conflicts occur when two or more activated package extensions describe the same project components. For example, if the Extension package extension adds a *submit* action to a form, then you will see a resource conflict if another package extension (for example, called Demo) also adds a submit action to that form. The submit action on that form can only be described by one package extension at a time.

Resolving Resource Conflicts

To resolve a resource conflict, you can either deactivate the package extension with the conflicting project component or you can delete the project component creating the conflict from one of the package extensions. Continuing the example from above, you could either deactivate the Demo package extension or you could delete the submit action from the Demo package extension.

Dependency Conflicts

Dependency conflicts occur when you change a project component in one of the packages that is not listed as a dependency for your current package extension. For example, if the Demo package extension is solely dependent on the common package, then the Demo package extension cannot make changes to the portal package without creating a dependency conflict.


Resolving Dependency Conflicts

To resolve a dependency conflict you can either add a dependency to the package extension, or you can move the changes to another package extension with the proper dependencies. Continuing the example above, you could either make the Demo package extension dependent on the portal package or you could move the changes from the Demo package extension to another package extension such as Extension, which is already dependent on the portal package.

Viewing Conflict Information

The Information on Selection tells you whether you have a resource or a dependency conflict.

To view conflict information:

- 1 Select a node with an exclamation point  icon displayed next to the name from the Project Explorer view.
- 2 Click View > Information on Selection.

A new information window will be displayed at the bottom of the Studio interface. This window displays information on the conflict.

You can do these from any computer with network access to your Web application installation and a copy of Studio installed.

For additional information about a particular Web application component and its possible settings, refer to the *Studio Introduction* and the Studio online help.

Deploying to UNIX Platforms

Using Studio's FTP connection feature, you can deploy tailored Web applications from a Windows machine running Studio to a UNIX platform. This feature will push the deployment files to a remote machine using an FTP connection.

Requirements

To use this feature you must have the following components:

- An FTP server on the target UNIX machine.
- A user name and password for the FTP connection.

Note: After the files are on your UNIX system, you will need to copy the files from the FTP home directory to an existing Web server virtual directory.

- An open network connection from the Studio source machine to the target UNIX machine.

Configuring for FTP Deployment

To configure Studio to FTP deployment files to a remote machine:

- 1 From Studio, click Build > Project Settings.
- 2 Click the FTP Connection commands tab, and then enter the following settings:
 - Host Name—the fully qualified domain name of the target machine to which you want to open to an FTP connection.
 - Port—the FTP port connection to use. The default value is port 21.
 - User Name—the user name to be used with the FTP connection.
 - Password—the password to be used with the FTP connection.
- 3 Click OK to save the FTP settings.

Deploying via FTP

To deploy files via FTP:

- 1 From Studio, perform an Integral build of your project to build all project files.
- 2 Click Tools > Deploy on FTP to transfer files to the remote machine.
Studio deploys files to the home folder of the user account you configured in Project Settings.

International Builds

The languages available are determined by the Web application you have installed.

Peregrine OAA Platform 2.2 supports the following languages:

- English
- French
- Italian
- German
- Spanish
- Japanese
- Polish

Configuring for an International Build

Follow these steps to create and display international builds of Peregrine Web applications.

Configure Studio

- 1 Set the font to support the intended build language(s).
 - a Click Tools > Options.
 - b Click the Appearance node on the options tree.
 - c Use the Language selectbox to choose the language in which you want Studio to display information.
 - d Use the Property Editor and Multi-language Text Editor selectboxes to choose the font names associated with the Studio Text Editor.
 - e Click OK to save your new settings.
- 2 Set the Build Options to create an International build and select the languages for which international strings will be built.
 - a Click Build > Project settings.
 - b Click the Build Options tab, and then click the International build checkbox.
 - c Select the checkboxes next to the language strings you want to include in your international build.
 - d Click OK to save your new settings.

Configure the Web application

- 1 Define the languages supported using the Admin module.
 - a Log in as an administrator (the administrator login page is located at `admin.jsp`).
 - b Click Settings.
 - c On the Common tab, enter the two letter ISO-639 language code for the languages you want to support in the Locales field. The first code entered will be the default language used. The other languages you define will be available in a drop-down list.
 - d In the Content type encoding field, enter the character encoding of the JSP files from the following table.

Character Encoding	Character Set
ISO-8859-1	U.S. and Western European character sets. This is the default character set used by Studio.
Shift_JIS	Japanese character set
ISO-8859-2	Polish character set

- e Click **Save** at the bottom of the Settings form to save your changes.
 - f On the Console form, click **Reset Server** to implement your changes.
- 2 Select the display language at login.
 - a Log in from the user login page (`login.jsp`).
 - b Use the Language drop-down selectbox available underneath the User Name and Password fields to select the display language for this session. The languages in this list are determined by the languages listed in the Locales setting defined above.

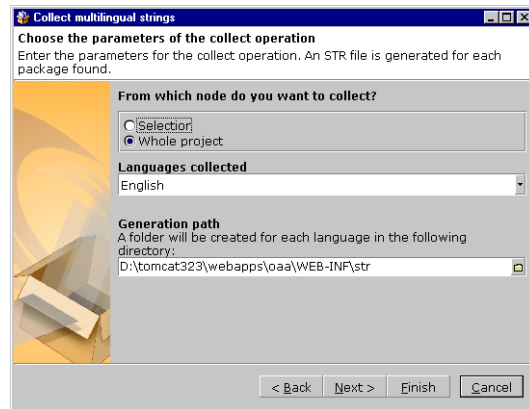
Exporting Strings for Translation

After you have completed the configuration steps described above, you are ready to export the strings for translation and create an international build of your Web application.

To export strings for translation:

- 1 From Studio, click Tools > Collect multilingual strings. This will start the String Collector wizard.

- 2 Click **Next** to go to the first page of the wizard.



- 3 Enter the following settings:
 - From which node do you want to collect?—select the Selection option to collect strings from the current node in the Project Explorer tree, or select Whole project to collect strings for all nodes in the project.
 - Languages collected—select the source language of the strings you want to export.
 - Generation path—enter or use the folder icon to browse to the location where you want the string file to be created. The string file for each language will be saved under a separate folder. The folder name will be the ISO code for the language you selected in the Languages collected field.
- 4 Click **Next** to go to the next page of the wizard. Click **Finish** to export string files from your project.
- 5 Click **Finish** again to close the wizard.

To translate string files:

- 1 Open the string file to be translated in a text editor or translation program. The string file uses the format illustrated below:

```
Studio_Address, "source string"
```

For example, if you had a form with an address of EMPLOOKUP_EMPLOYEELOOKUP_SEARCH_LABEL, the string for the label of this form would be:

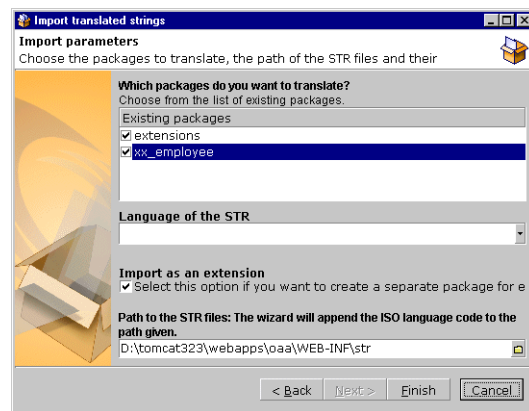
```
EMPLOOKUP_EMPLOYEELOOKUP_SEARCH_LABEL, "Search"
```

- 2 Change the "source string" portion of the string to the target language of your translation. For example, to change the string listed above to French, enter the following:
EMPLOOKUP_EMPLOYEELOOKUP_SEARCH_LABEL, "Recherche"
- 3 Save the new string file in the str folder for the target language of your translation. This folder must use the default character encoding for the target language.

Importing Strings for Translation

To import translated strings into a Studio project:

- 1 From Studio, click Tools > Import translated strings. This will start the Imported translated strings wizard.
- 2 Click Next to go to the first page of the wizard.



- 3 Enter the following settings:
 - Which packages do you want to translate? Select the check boxes next to the names of the packages you want to import strings into and clear the checkboxes of packages you do not want to import translated strings into.
 - Language of the STR—select the target language of the translated strings you want to import.
 - Import as an extension—select this option to create a separate package for each language.
 - Path to the STR files—enter or browse to the location where the translated string files are located.

- 4 Click **Next** to go to the next page of the wizard. Click **Finish** to import translated string files into your project.
- 5 Click **Finish** again to close the wizard.

Adding to an Existing Frameset

You can add your Peregrine Web application to an existing frameset to incorporate into your corporate intranet. To do this, you will need to edit a JavaScript file within your project file and add a reference to your Peregrine Web application to the parent frameset.

To add a Peregrine Web application to an existing frameset:

- 1 Open the following file in a text editor:

```
<tomcat installation>\webapps\oaa\js\setDomain.js
```

or locate the file in the equivalent directory in your application server.

- 2 Add the following line to the bottom of the script:

```
setDomain(server name);
```

where **server name** is the name of the server where the parent frameset is located.

- 3 Save the file.
- 4 Add the following line to each JSP file that will include your Web application in a frameset. These files must be saved on the server listed in step 2.

```
<script language="JavaScript" SRC="js/setDomain.js">  
</script>
```

- 5 Save the updated JSP files.

4 Forms and Form Components

CHAPTER

This chapter provides an introduction to the components of a form in your Web application and how to tailor a form.

Each page displayed in your Web application consists of a form and several form components. Each form also has the following supporting elements:

- An onload script that gathers the data that the form displays or processes information from the previous form.
- A schema, which maps to fields in the database and determines what information to display.


For a complete list of each component available in Studio, see Appendix A, beginning on page 149.

Tailoring Forms

You can change a form's title, instructions, onload script, and component labels. You can also hide a form component and make a form read-only.

To tailor Web application forms:

- 1 Open the project file you want to tailor in Studio.
- 2 Select or create a package extension in which to save your changes.
- 3 Open your browser and log in to your Web application.
- 4 Navigate to the form you want to tailor by doing one of the following:
 - Click the Studio address in the Form Information banner of the Web application form. Studio will appear as the active window and display the current form's properties page.OR
 - In Studio, locate the form in the Project Explorer.
- 5 Make your changes to the Web application form in Studio.
- 6 Save the project file.
- 7 Rebuild the project file.

Tip: If you have only made changes to one or more forms in an activity or module, use the Differential Build option () to build just the components that have changed. This option will reduce the time needed to build your Studio project.
- 8 Refresh the browser to reload the form you modified.
- 9 Review your changes and test the added functionality.

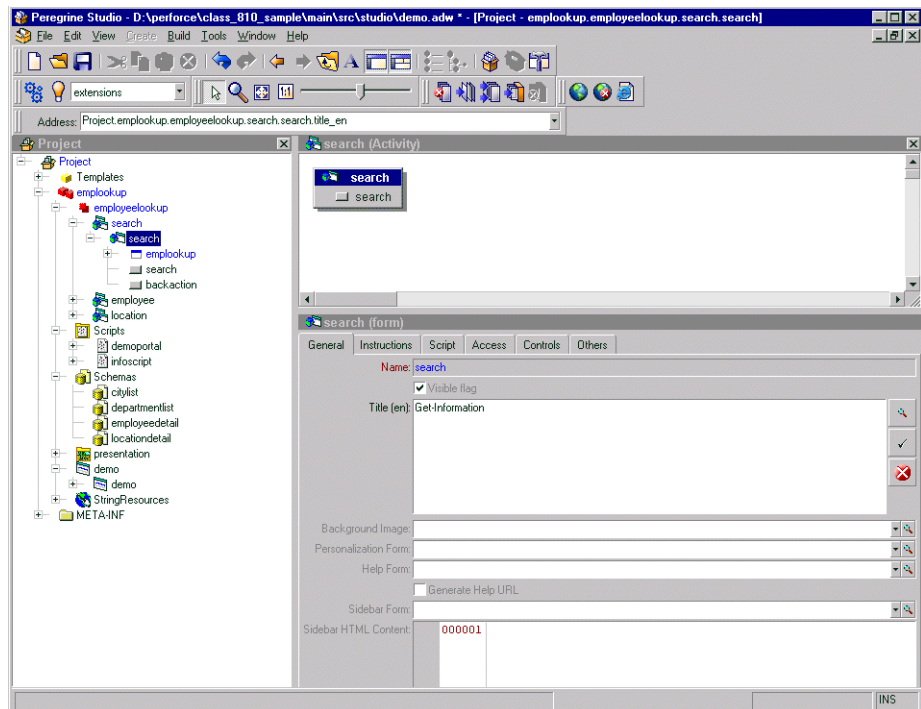
Tip: If you want to test new access right settings for your components, log on to the Peregrine Portal with several different users with different access rights.

Changing Form Titles

Each Web application form can show a title at the top of the frame. If you want to change or remove the title displayed for a particular form, set the following form properties.

To change a form title:

- 1 Open the form's properties in Studio.
- 2 In the Title (en) field, enter the new form title
- 3 Click the check mark button () at the right of the field to accept the new title.
- 4 Save and build your project file.

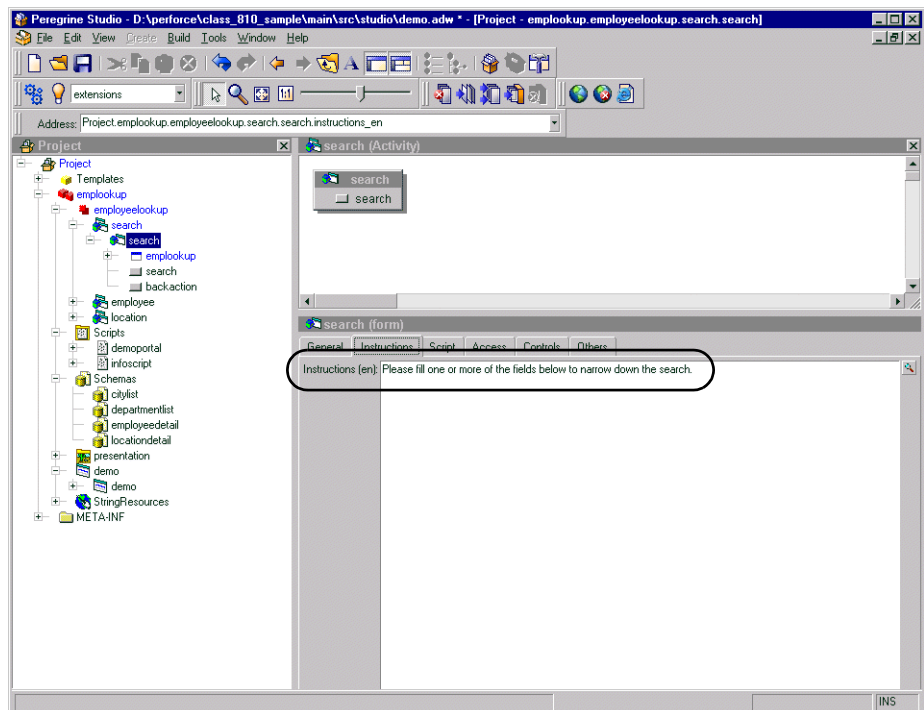


Changing Form Instructions

Most forms display a set of instructions at the top of the frame. You can change the instructions to match any changes you make to the form's interface.

To change form instructions:

- 1 Select the form in the Project Explorer.
- 2 Select the Instructions tab in the Properties window.
- 3 In the Instructions (en) field, enter the new form instructions.
- 4 Click the check mark button () at the right of the field to accept the new form instructions.
- 5 Save your project file.
- 6 Build your project file.



Changing a Form's Onload Script

A form's onload script gathers all the data that the form displays, or processes information from the previous form. Many onload scripts also invoke schemas to present back-end database information in a format that is easier to map to particular form fields or form components.

Typical tailoring tasks include changing the script launched when a form is loaded or directly editing the onload script itself. You can perform both of these tasks from Studio.

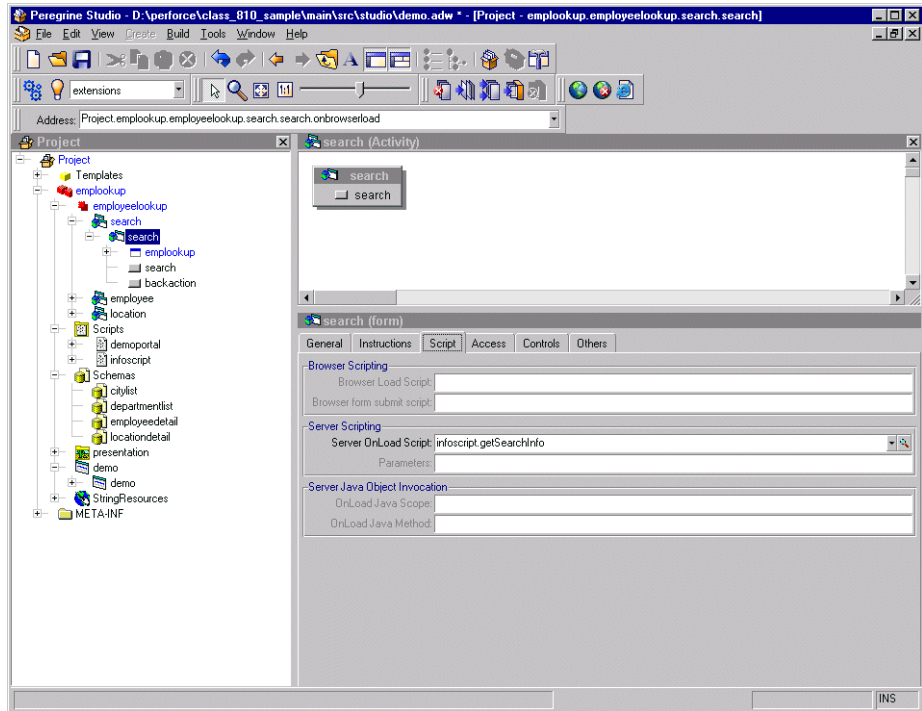
To change the onload script invoked by a form:


- 1 Select the form in Studio.
- 2 Click the Script tab in the Properties window.
- 3 In the Server Onload Script field, enter or select the script you want to invoke when this form is loaded. You can use the drop-down list to select any of the scripts saved in your project file.
- 4 Save your project.
- 5 Build your project file.

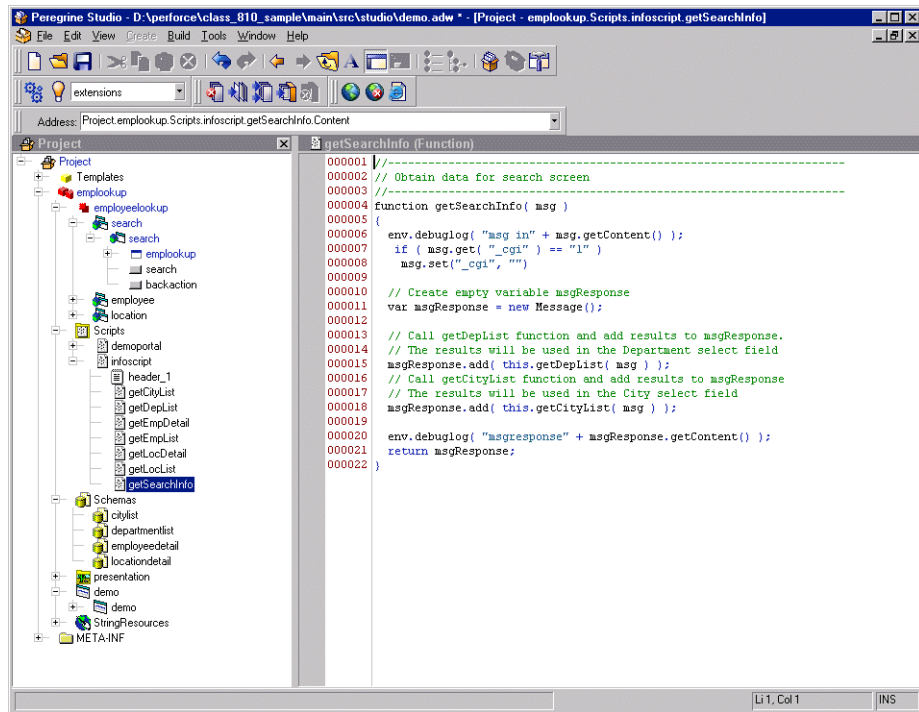
To edit the onload script:

- 1 Select the form in the Project Explorer.

2 Click the Script tab in the Properties window.



- In the Server Onload Script field, click the magnifying glass button () to view the script in the Studio text editor.



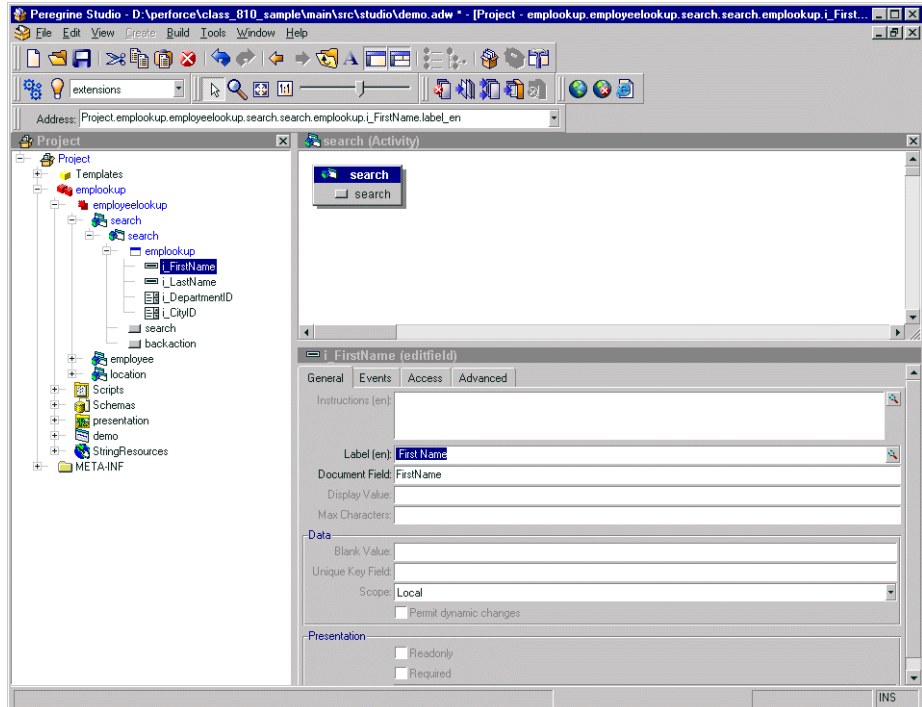
- Make any changes to the script in the text editor.
- Save your project.
- Build your project file.

Changing Form Component Labels

Many form components contain a label that is displayed next to or above the form component. Some of the most commonly configured form components are the field form components (check box, select box, edit field, and so forth).

To change a component label (field label):

- 1 Select the form in the Project Explorer.
- 2 On the General tab, select the Label (en) field, enter the new form component label, and press ENTER.
- 3 Save your project.
- 4 Build your project file.

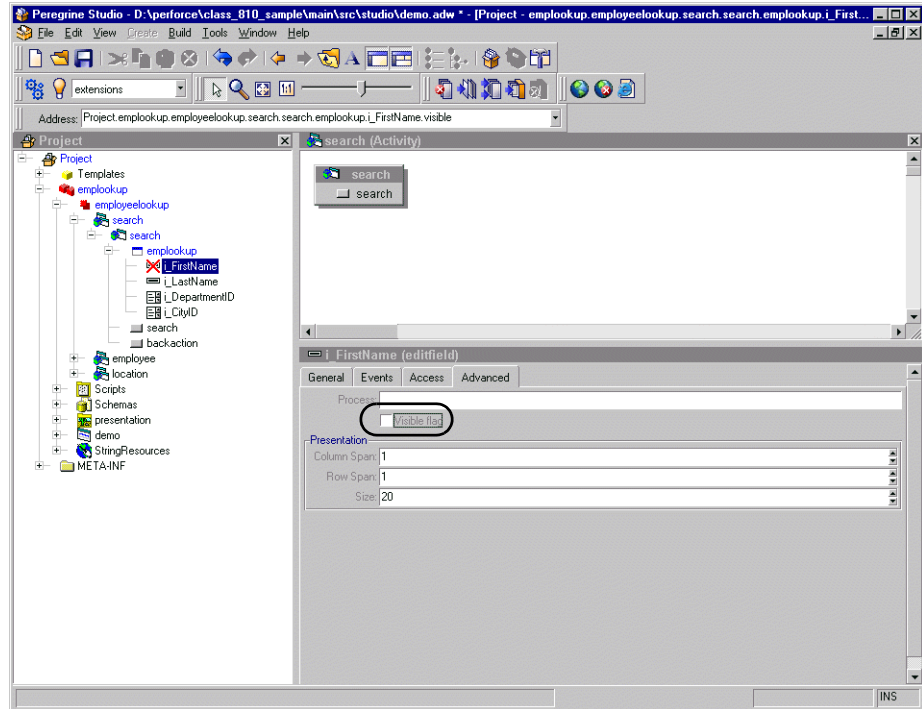


Hiding Form Components

All form components have a Visible flag property that hides or displays the component in the Web application interface. If you want to remove a form component from the interface but still have it available in Studio, you can toggle the form component's Visible flag to No. This prevents the form component from being part of the next Studio build. Non-visible (and thus non-built) form components are displayed with a red X over the form component icon in the Project Explorer tree.

To hide a form component in the interface:

- 1 Select the form in the Project Explorer.
- 2 On the Advanced tab, clear the Visible flag option.
- 3 Save your project.
- 4 Build your project file.



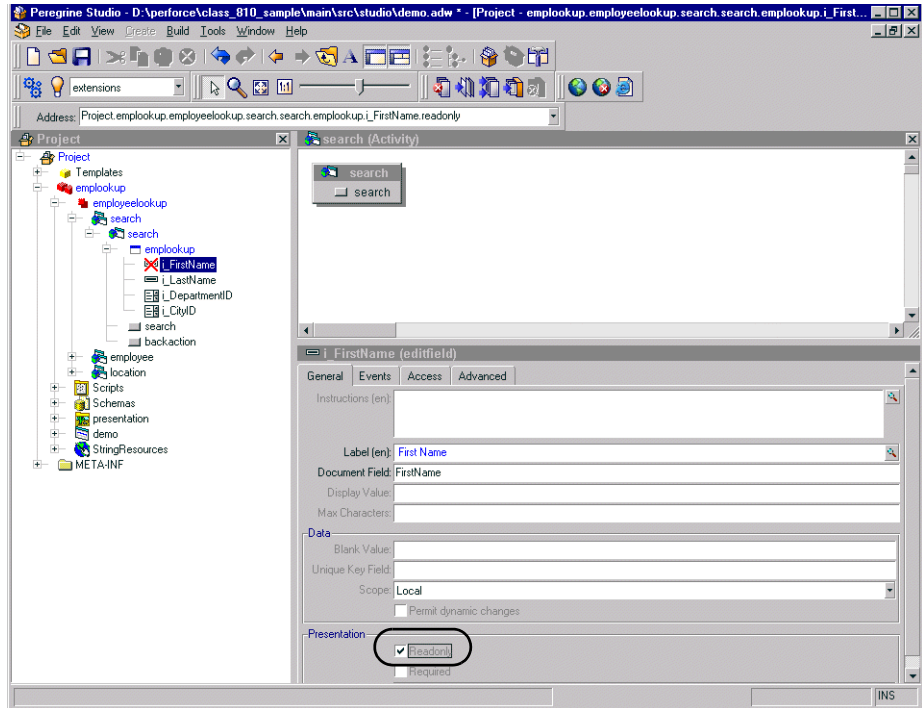
Changing a Form Component to Read-only

Certain form components such as edit fields and text areas are available for users to enter and change information. If you want to restrict these form components so that they only display data, you can set the readonly attribute for the form component. The data displayed by a readonly form component will no longer have a bounding box or area to indicate that it can be edited or changed.

You can change a form component back to its original state by removing the readonly attribute.

To make a form component read-only:

- 1 Select the form in the Project Explorer.
- 2 On the General tab, select the Readonly check box.
- 3 Save your project.
- 4 Build your project file.



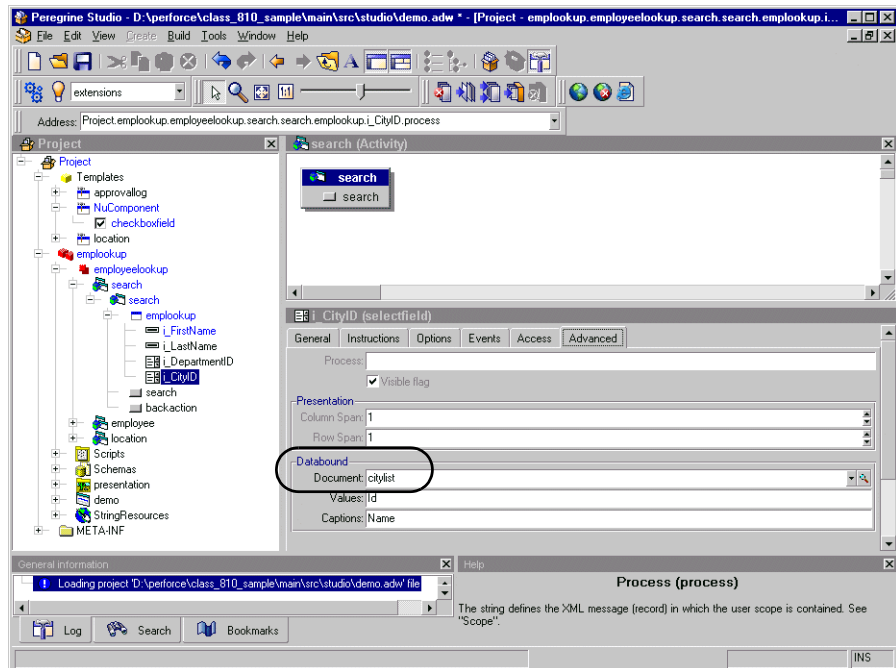
Changing the Schema that a Form Component Uses

Certain form components such as selectfields and simple tables use a schema to determine what information to display. You can change the information these form components display by changing the schema defining the document fields. In some cases you may also need to change other form component attributes that depend on the fields defined in the schema.

To change the schema that a form component uses:

- 1 Select the form in the Project Explorer.
- 2 Click the Advanced tab.

- 3 In the Databound section, Document field, enter or select the name of the schema that you want to use as the source document for this form component.



- 4 Save your project.
- 5 Build your project file.

Changing the Document Field that a Form Component Uses

Certain form components such as selectfields and table columns use a particular document field of a schema to determine what information to display. You can change the information these form components display by changing the document fields these components use.

Note: The list of document fields available to a form component is determined by the schema used. Studio does not validate the document field you select.

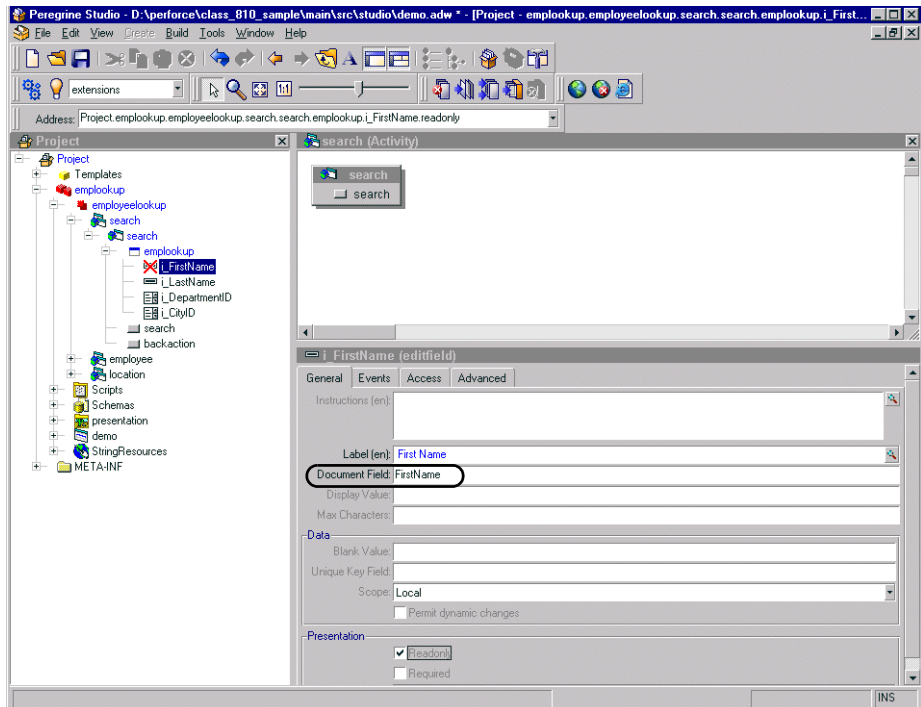
To change the document field that a form component uses:

- 1 Select the form component in Studio to display the component's properties.

- In the Document Field field, enter the name of the field in the XML message where this form component's information is stored.

Note: The field you select must be defined as an attribute in the schema defined in the form component's properties.

- Save your project.
- Build your project file.



Specifying a Document Field Name

The Document Field attribute of forms is always mapped to an element in the Message object returned by the form's Onload script.

The Archway servlet formats Message objects as XML files using the tag definitions and back-end database table and field information that the schemas provide.

The Document Field attribute of a form component must map to an <attribute> element in a schema.

You can specify the Document Field attribute that a form component uses in one of several ways:

- If the Document Field attribute has a unique <attribute> name in the schema, you can list just the <attribute> name.
- If the Document Field attribute is repeated in the schema, you must specify the nested <document> name or names and the <attribute> name. The <document> name and the <attribute> name must be separated by a slash character (/).
- If the Document Field attribute is part of a nested <document> element, you have the choice of either listing the <attribute> name by itself or specifying the some or all of the path using the syntax of <documents>/<document>/<attribute>. This syntax allows Web application developers to specify as much or as little of the document path as is needed to create a field attribute mapping.

Example

Suppose you are creating a form where users can review and submit asset requests. A typical asset request may be formatted as the following XML message:

```
<request>
  <Number>012345</Number>
  <Purpose>Asset Management</Purpose>
  <EndUser>
    <FirstName>Michaela</FirstName>
    <LastName>Tossi</LastName>
  </EndUser>
  <Requester>
    <FirstName>Richard</FirstName>
    <LastName>Hartke</LastName>
  </Requester>
</request>
```

In this case, the <FirstName> and <LastName> tags are repeated in two different sections of the XML message. To display these tags in a form, you will need to specify more of the document path when you enter the path of the Document Field attribute. The entries below illustrate the minimum document path needed for the Document Field attribute in a form component.

Number
Purpose
EndUser/FirstName
EndUser/LastName
Requester/FirstName
Requester/LastName

You can also specify the Document Field attribute path using all the elements of the XML message. The following entries illustrate the full document path that can be used for the Document Field attribute in a form component.

request/Number
request/Purpose
request/EndUser/FirstName
request/EndUser/LastName
request/Requester/FirstName
request/Requester/LastName

The number of elements that you must specify in the document path is determined by how you set up your schemas.

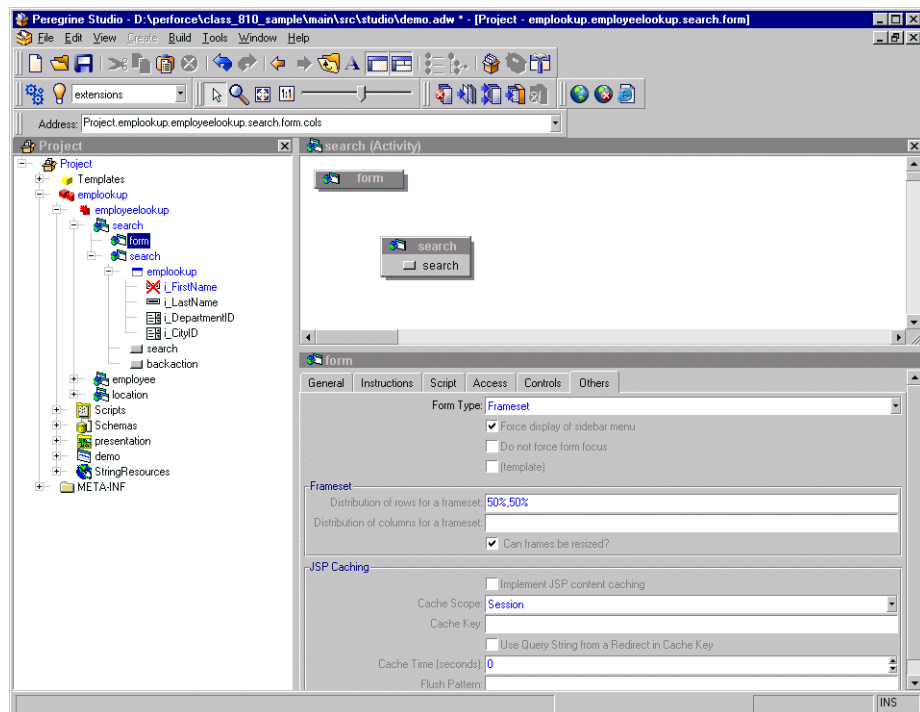
Displaying Forms within a Frameset

You can display forms within multiple frames by creating a special frameset form. All frames within a frameset form will be displayed within the frame normally reserved for forms.

To display forms within a frameset:

- 1 Right-click the activity where you want the frameset form to be, point to New, and then click Form.
- 2 Click the Others tab.
- 3 Select Frameset from the Formtype drop-down list box.
- 4 Enter row and column sizes in the Frameset pane.

Note: You can use percentage to describe frameset size properties.



- 5 Create a new form for each frame in the frameset form.
- 6 Create a redirection under the frameset form for each target form in the frameset.

- 7 Save your project.
- 8 Build your project file.

To display the form title within a frameset:

- 1 Open the frameset form's component properties in Studio.
- 2 Create a new server onload script within your project.
- 3 Add the following lines to the script:

```
top.setTitle(TitleText)
```

Where *TitleText* is the title you want to display at the top of the frameset.
- 4 Open the component properties page for the target form within the frameset.
- 5 Click the Script tab.
- 6 Select the server script you created in step 2.
- 7 Save your project.
- 8 Build your project file.

Types of Form Components

For a complete list and description of all Studio components, see Appendix A, beginning on page 149.

Component Template Containers

The component template is a special type of container used to store groups of preconfigured form components. The form components stored in a component template can be reused through your project. After you create a component template, the component template name appears in the templates list of the Create and New menus. A component template references all the child form components and attributes settings defined in the template.

If you add a component template to a Web application and do not modify it, Studio saves the form components as links to the component template. If you make changes to the form components in the template, Studio saves the changes you have made and links to the components that you did not change.

Tip: You can use component templates to save the common elements of your forms. For example, if several of your forms contain search functionality, then you could create a component template that automatically calls the correct search schema, queries your back-end system, and displays the proper search fields.

To create a component template:

- 1 Right click the Templates nodes and go to New > Field Container > Component Template.
Studio adds a new component template node to the Project Explorer Tree.
- 2 Enter the name for the component template.
- 3 Right click the new component template node and use the New option to add form components.
- 4 Configure the form components you add to the template component. Studio uses these settings as the default settings of the template component.
- 5 Save and build your Studio project.

The new template component will now be an option in the New menu.

Note: Do not copy and paste or drag and drop items between template components. Instead add form components via the context-sensitive or Create menus. Studio does not use the linking features of template components on items that you copy from existing template components.

To add a component template to a form:

- 1 Right-click the form where you want the component template to be.
- 2 From the New menu, select the template you want to add.

Form components you can add to a component template

- ▶ All except Action and Redirection.

Tip: You can use a component template as the container for any form components that require a container. This is typically done for form components such as hiddenfields where you are not concerned about the display of the fields.

Attribute categories you can set for a component template

- ▶ Name, Access rights, and Geometry.

Fieldsection Containers

The fieldsection is a container that aligns fields into a column. The fieldsection displays each field on its own line in the column. The fieldsection also aligns the field labels along the left of each field. Each fieldsection can have a border that surrounds the columns and visually indicates that the fields in the container are related. You can also add a header or instructions to your fieldsection as well as add labels and instructions to the individual fields in the fieldsection.

Tip: You can use the fieldsection form component to group and align related input fields. For example, if you have several fields to input search information, you can align the fields in a single fieldsection and add a header and instructions that will apply to all fields.

To create a fieldsection:

- 1 Right click the form where you want the fieldsection to be.
- 2 Go to New > Field Container and click Field section.

Form components you can add to a fieldsection

- ▶ Field, HTML, Header, and Instructions.

If you select the Header or Instructions form components, Studio will display the text editor screen for you to enter HTML code for your header and instructions. Studio will not check the validity of your HTML code.

Attribute categories you can set for a fieldsection

- ▶ Name, Access rights, Geometry, and Presentation.

If you plan on having multiple fieldsections in a form, you can use the border Presentation property to display a line around a fieldsection to help visually distinguish the fieldsection from other elements in your Web application interface. You may also want to add a Form Columns layout container to display your fieldsections in two or more facing columns rather than a single column down the form.

Text Edit Fields

A text edit field provides a bordered field in which to display or enter a value as plain text. Text edit fields can only be added to forms within a container such as a component template or fieldsection.

Peregrine Web applications use text edit fields to provide a space for user keyboard input. The text edit field saves the text entered into a particular schema field when a user submits the form.

Tip: To use a text edit field for text input, add an action to the form that submits the field information to another form. Set the Document Field attribute of the text edit field to the corresponding attribute name used in the document schema.

You can also use text edit fields to display information. To display information in a text edit field, create an onload server script that performs a document query, and then map the text edit field to one of fields of the schema.

Tip: To use a text edit field for display of information only, add a schema to the parent form that defines the information to be displayed. Set the Document Field attribute of the text edit field to the corresponding attribute name used in the schema. Set the readonly attribute under Presentation to Yes if you do not want users to change the information displayed.

To create a text edit field:

- 1 Right-click the container where you want the field to be. This displays the context-sensitive menu.
- 2 Go to New > Field and click Text Edit.

Form components you can add to a text edit field

- ▶ None.

Attribute categories you can set for a text edit field

- ▶ Name, Access rights, Advanced, Data, Events, Geometry, Presentation, and Strings.

Selectbox Fields

A selectbox provides a drop-down list box from which users can select predefined values. You can add items to the selectbox in one of two ways:

- Explicitly define the options. The selectbox always displays the options you enter and always displays them in the order they are defined in by the Geometry Order attribute.
- Query a back-end system and generate an XML document. The selectbox displays options gathered from your database and the schema used to generate the XML document. Typically, the selectbox uses the same schema as that used by the parent form of the selectbox. If you use a schema to display the items in the selectbox, then the Document field attribute you enter must match the corresponding attribute name defined in schema.

Tip: Use the schema query method to avoid duplicating information that is already stored in your back-end databases. If you explicitly enter the options in the selectbox, then you have to update, rebuild, and re-deploy every time you change the list of selectbox options. If you store the selectbox options on your database, then you only need to change the database values, and your schema query automatically picks up the changes.

When you are working with selectboxes, keep in mind that:

- You can only add selectbox fields within a container such as a component template or fieldsection.
- Users cannot add entries to selectbox fields. To implement such functionality, Web application developers would have to write a client-side JavaScript to allow updates to the selectbox.

- Web applications use selectbox fields to constrain user input to a list of predefined items. The selectbox field saves the selected item to a particular field when a user submits the form. The field used to save the information must match a field defined in a document schema.
- If you have a large number of selections for users to choose from you may want to use a lookupfield in place of a selectbox. The advantages of lookupfields are that they can be personalized and that they do not require page loading until the lookupfield is selected, which reduces the amount of time necessary to render the form.

To create a selectbox field:

- 1 Right click the container where you want the field to be.
- 2 Go to New > Field and click Selectbox.

Form components you can add to a selectbox field

- ▶ Option. The Option form component allows you to explicitly define the entries displayed in the selectbox.

Attribute categories you can set for a selectbox field

- ▶ Name, Access rights, Advanced, Data, Databound, Events, Geometry, Presentation, Strings.

Databound attributes

The Databound attributes are where you will define what schema and schema attributes provide the information for the selectbox. The following list describes what information to enter in the Databound attributes.

- Captions. Enter the attribute name from your schema that defines the information you want displayed in the selectbox.
- Document. Enter the schema name you want to use to query and display the information requested in the selectbox.
- Values. Enter the attribute name from your schema that defines what information you want to use to sort and identify the information in the selectbox. This value can be identical to the displaylist attribute, but it is recommended that you use the Id attribute name defined in the schema. The Id attribute is the preferred choice because it is a unique value and requires less memory to sort since it is just a number.

Hidden Data Fields

A hidden data field stores form information without displaying it to the user. The information stored in a hidden data field is passed to other forms when the form is submitted.

Tip: You can use hidden data fields to prevent users from having to input the same information on multiple forms. For example, if a user enters contact information in one form, then you can use hidden data fields to store this contact information in later forms.

To create a hidden data field:

- 1 Right click the container where you want the field to be.
- 2 Go to New > Field and click Hidden Data field.

Form components you can add to a hidden data field

- ▶ None.

Attribute categories you can set for a hidden data field

- ▶ Name and Data.

Redirections

A redirection takes users to another form when the onload server script generates a certain condition. A conditional redirection requires the parent form to run a server script when it is loaded. The server script must check for a particular condition and output a condition message when this condition occurs.

You can only add a redirection to a form; you cannot add a redirection to a form component.

Tip: You can use a redirection to take users to a form when they enter particular information or a particular result, such as an error or no results, is generated.

To create a redirection:

- 1 Right-click the form where you want the redirection to be. The context-sensitive menu is displayed.
- 2 Go to New and click Redirection.

Form components you can add to a redirection

- ▶ None.

Attribute categories you can set for a fieldsection

- ▶ Name, condition, Access rights, Link Parameters, Presentation, and Strings.

Redirection attributes

For most redirections, the two most important attributes to set are the condition and the target form.

- Condition. Enter the message that your server script generates when a user should be redirected to another form. If there is no condition, the redirection will take effect every time the page is loaded.
- Target form. Enter the full Studio path to the form where the user should be redirected.

Table Form Components

A simple table is a container to display information generated from a schema document query. The simple table form component only has two basic functions by itself. The simple table form:

- Calls the schema that will generate the table data, and
- Describes how the data will be displayed in the child columns of the table.

A simple table requires child columns to actually display data.

To create a simple table:

- 1 Right-click the form where you want the table to be.
- 2 Go to New > Table and click Simple Table.

Form components you can add to a simple table

- ▶ Link, Text Column, Entry Column, Spinner Column, Select Column, Radio Button Column, Checkbox Column, Image Column, Link Column, and Lookup Column.

Attribute categories you can set for a simple table

- ▶ Name, Document, Geometry, Dynamic Columns and Headings, Access rights, Columns, Links, and Advanced.

The Document attribute defines the schema the simple table uses. You can enter a schema name or select one from the drop-down list box.

Simple tables include a built interface to view large tables in smaller pages. You can use the size attribute in the Geometry section to set the number of rows to display on one page. When users want to view more of the table results, they can click on the next x rows button to view the next page of table rows. All simple tables will include the link icons to browse forward and backward in the table.

Table Links

A table link allows the user to click on a table row and be redirected to another form. The table link also saves some field information about the row the user selects and submits this information to the target form. Table links are typically used for two functions:

- To display more information about an item selected in the table, or
- To copy certain information about the item selected in the table into a new form such as, for example, the price of an item in a purchase request form.

To create a table link:

- 1 Right-click the table where you want the table link to be.
- 2 Go to New > Link and click Table Link.

Form components you can add to a table link

- ▶ None.

Attribute categories you can set for a simple table

- ▶ Name, Access rights, Data, Link Parameters, Presentation, and Strings.

Table link attributes

For most table links, the two most important attributes to set are the Document field and the target form.

- Document field. Enter the field that describes what information should be passed when a table link is submitted. The Document Field attribute should match the attribute name of an item in your schema. The attribute is typically set to the Id schema attribute.
- Target form. Enter the full Studio path to the form where the user should be redirected when they click on a table row.

Text Columns

A text column displays the results of a document query in plain text. Each text column displays one field of information from a back-end database. The field must match an attribute name listed in the document schema of the parent table.

When working with text columns, keep in mind that they:

- Are always read-only and cannot be used to update information in the back-end database.
- Can only be added as child nodes of a simple table.

To create a text column:

- 1 Right click the table where you want the text column to be.
- 2 Go to New and click Text Column.

Form components you can add to a text column

- ▶ None.

Attribute categories you can set for a text column

- ▶ Name, Access rights, Data, Geometry, Image, Presentation, and Strings.

Text column attributes

For most text columns, the two most important attributes to set are the Document field and the Label_en.

- Document Field. Enter the field that describes what information should be displayed in the text column. The Document Field attribute should match the attribute name of an item in your schema.
- Label_en. Enter the label you want displayed in the first row of the table as the column heading. If you are using dynamic headers and columns, you will want to leave this attribute blank.

Actions

An action displays a button on the form to submit form information or follow a particular link. The following is a list of the possible actions you can include in your forms:

- Action. Use to submit form information or follow a link.
- Back. Use to navigate back to the previous form.
- Close. Use to close pop-up windows.

- **Default Action.** Use to define a form's submit action when no buttons are displayed in a form.
- **Home.** Use to navigate to the portal home page.
- **Print.** Use to print the current form.

To create an action:

- 1 Right-click the form where you want the action to be.
- 2 Go to New > Action and click the action type you want to add.

Form components you can add to an action

- ▶ None.

Attribute categories you can set for an action

- ▶ Name, Access rights, Image, Link Parameters, Presentation, and Strings.

Action attributes

For most actions, the three most important attributes to set are the Image Folder, Target form and the Label_en.

- **Image Folder.** Enter the file name of the image to be used for the button.
- **Target form.** Enter the full Studio path to the form where the user should be redirected when they click on the button.
- **Label_en.** Enter the label you want displayed in the button.

5 Adding Personalization Functionality

CHAPTER

Personalization of Peregrine Web applications is provided in two ways:

- Personalization is available to end users in Peregrine Web applications that have been built using Document Explorers (DocExplorers). Authorized users can change the appearance and functionality of certain Web applications directly from the application interface.
- Pages with Personalization capabilities can be added to Web applications by creating new DocExplorers. This functionality can be enabled only by using Peregrine Studio.

Supporting Personalization

To add Personalization capabilities to your Web application, you must have these components:

- An AssetCenter or ServiceCenter back-end database, or a back-end system that uses BizDoc, such as Oracle. Personalization requires you to store users' login rights and personalization changes in one of these databases.
- A user account with Personalization rights enabled. A user's login profile determines the level of Personalization rights the Web Application grants a user. Users' Personalization rights determine not only what personalized components they can see and change, but also determine whether other users will see their personalization changes. A Peregrine Web application may display some personalization changes for an entire organization or specific user groups, but it may also limit the display of changes to a specific user.
- A schema that supports Personalization. There are a number of personalization-specific schema entries that determine how Studio displays fields in personalized forms.
- A configured DocExplorer activity for each Web application for which you want to provide Personalization. Each DocExplorer activity must have an adapter name and a schema. A DocExplorer can only use one schema at a time.

Activating Personalization

Access to Personalization features is defined using the `getit.personalization` access right/capability word stored in the Web application database. You can define the scope of end users' personalization rights by selecting one of the options from the End User Personalization options on the Settings page of the Admin module.

The End User Personalization setting can have one of three values:

- Disabled. End-users see only the default Personalization settings made by the administrator. This setting prevents any further personalization from being made by non-administrators. Your Web application does not display the Personalization wrench icon to end-users.

- Limited. End-users will inherit the default Personalization settings made by the administrator, but they can also make limited changes to the Web application interface. This setting allows end-users to add or remove only the fields that were originally included in the administrator's default personalization. This setting also prevents end-users from changing read-only fields to editable fields.
- Enabled. End-users will inherit the default Personalization settings made by the administrator, but they can also make substantial changes to the Web application interface. This setting allows end-users to add or remove any field listed in the schema.

You can also grant users administrator rights by adding the `getit.personalization.admin` access right/capability word to the user profile stored in the Web application database. Administrator users will have the following additional rights:

- Create. A Create button is displayed in search forms that enables users to create new entries in the back-end database.
- Update. An Update button is displayed in detail forms that enables users to change the information that is listed for entries. Changes are also submitted to the back-end database.
- Delete. A Delete button is displayed in detail forms that allows users to delete records from the list of entries. The record is also deleted from the back-end database.
- Settings will be inherited by other users. This settings enables all personalization changes made by an administrator to be inherited by personalization end-users.

After you set up a Studio project to use Personalization, some or all interface changes and additions can be accomplished using the Personalization interface. Typically, an administrator or user with administrative rights will personalize the interface for all other users. If you want the personalization changes from one user to be inherited by other users and groups, you will need to set up a hierarchy of users.

Personalization Hierarchies

Personalization changes cascade down to other users based on the personalization hierarchy defined in the `personalize.getHierarchy` ECMAScript. By default, this script assigns users with `getit.admin` or `getit.personalization.admin` rights to the `/admin` group. Changes made by members of the `/admin` group cascade down to all other users.

If a user does not have the admin rights described above, the script will assign the user to a unique group called `/admin/user name`. Such users will see all personalization changes made by administrators plus any personalization changes they have personally made.

You can change the default hierarchy of users by modifying the `personalize.getHierarchy` ECMAScript. The output of this script must meet the following criteria:

- The script output must begin with a slash character (`/`).
- Each hierarchy level must be separated with a slash character (`/`).

Personalizing with DocExplorers

DocExplorers allow Web application end users a means to create and customize searches of connected databases. From the end user perspective, a DocExplorer is a special activity that allows someone to personalize part of the interface. The user's profile determines the Personalization rights granted.

From the Web application developer's perspective, a DocExplorer is a template activity that allows for the rapid development of Web applications without the need to rebuild a Studio project for every change. A DocExplorer enables you to add or remove fields, change the layout of a form, and change interface elements such as headers and buttons in real time using the browser interface.

DocExplorer Forms and Functions

All DocExplorers provide the following forms and functionality:

- A search form for defining search criteria.
- A list form for displaying search results.
- A details form for displaying detailed information about search results.

By default, only a user with `getit.admin` user rights can personalize all DocExplorer forms. Typically, this person will be an administrator who decides what document fields and layout will be available in the Document Explorer forms. End-users will then inherit the administrator's personalized forms. This may be the only form of Web application tailoring that is enabled at some organizations.

If you grant your end-users administrative rights, they can use Personalization for the following actions:

- Add a new record to the database.
- Update existing records in the database.
- Delete existing records from the database.

To use a DocExplorer, you must have defined a schema for the back-end database you want to query. Peregrine Web applications come with a set of fully functional schemas that you can use for your DocExplorers. You can put commands in a schema that will then be picked up by a DocExplorer. For example, you could define a field to be read only, such as a problem ticket number, or establish user access restrictions to certain forms.

Adding a DocExplorer Reference

A DocExplorer reference creates a link to a set of reusable JSPs. These pages provide functionality to search, display a list of data returned from the search, and ability to drill down into this data for more detail. Adding a DocExplorer reference is the procedure that will provide the functionality you need for most instances. You can then use Personalization to tailor the pages as desired.

To add a DocExplorer Reference:

- 1 Right-click on a Module component in your project. Select `New>DocExplorerReference`.
- 2 Include the shared templates package in the project as follows:

- a From the New menu, select **Add package to project**.
 - b Browse to the location of the shared templates file in the source code for your Web application. Select the file, and then click **Open**.
- 3 Expand the DocExplorerReference node.
 - 4 Select the redirect node, and then click the Link Params tab in the properties window.

The Param field has the following line filled in:

```
_DocExplorerDocument=<DOCUMENT_NAME>&_DocExplorerBackend=<TARGET_NAME>
```

Replace <DOCUMENT_NAME> with the schema name you want to use for this DocExplorer.

Replace <TARGET_NAME> with the alias that will designate the connection to an adapter. This setting should be defined according to the purpose of your DocExplorer. For example, if you added a DocExplorer reference to a request module in Get-Resources, using AssetCenter as the back-end system, you would set the target_name parameter to *ac*, as designated on the Get-Resources tab, Get-Resources Target field on the Admin Settings page. This setting could also be the Default capabilities setting on the Portal DB tab—*portalDB(getit.portal)*.

- 5 Save your project.
- 6 Click the **Differential build of project** button to rebuild your project.

Personalizing DocExplorer

After you have added a DocExplorer Reference, you can personalize the template pages.

To personalize your DocExplorer pages:

- 1 Log in to your Web application.
- 2 Click the activity name for your Document Explorer from the navigation sidebar. By default, the Document Explorer will be called DocExplorer. If this is the first time you have accessed the Document Explorer, the interface will display a blank search form.
- 3 Click the wrench icon on the upper right of the interface.
- 4 Make your changes to the search form, and then click **Save**.
Your personalized search form is displayed.
- 5 Click **Search** to display the results list form.

- 6 Click the wrench icon from the upper right of the interface.
- 7 Make your changes to the list form, and then click **Save**.
- 8 Click on any of the results displayed in your personalized list form to go to the detail form.
- 9 Click the wrench icon from the upper right of the interface.
- 10 Make your changes to the detail form, and then click **Save**.
- 11 If you have user rights to create documents, click the activity name for your Document Explorer from the navigation sidebar to return to the search form.
- 12 Click **Create** to display the create form.
- 13 Click the wrench icon from the upper right of the interface. Make your changes to the create form, and then click **Save**.

Adding Personalization to Lookup Fields


You can create automatically-generated lookup fields using Personalization. These personalization features reduce the number of forms and configuration necessary to create a pop-up window with lookup information. You can use Personalization features to configure two types of lookup fields:

- **Field Lookup**—use this lookup to select one particular field from your schema. For example, you might want to select just the Name field from your Employee schema.
- **Nested Document Lookup**—use this lookup to select one or more fields that are nested under a subdocument in your schema. For example, you could lookup the Location subdocument from your Employee schema to update several fields such as address, state, zip, and country.

Note: When you select an entry from a Nested Document Lookup, all the fields used by the lookup schema are returned. Any other form components that use these fields will be automatically updated. This allows users to quickly change multiple fields on a form.

To create a field lookup:


- 1 Right click the form to which you want to add the lookup, point to New, point to Field, and then click Lookup.
- 2 Enter the following settings for the Data attribute:

- Display Field—the name field that you want to be displayed in the Web application form when a user makes a selection from the lookup field. If you do not enter a value for this parameter it defaults to the Document Field parameter described below.
 - Document Field—the name of the key field used to uniquely identify each individual document record. The value of this field is used to lookup the document field defined in step 3 below. This value will also be posted to the onload script when a particular lookup entry is selected.
- 3 Enter settings for the following DocExplorer Adapter attributes:
 - Adapter—the name of the database adapter where your lookup information is stored.
 - Document Path—the name of the schema and field name that you want to lookup and enter into the Web application form. The naming convention used with this parameter is schema name.field name with a period (.) between them. For example, the entry employee.name will lookup the name field from the employee schema.
 - 4 Enter the following setting for the Link Parameters attribute:
 - Target Form—enter docExplorer.fieldlookup.start as the form name. This value enables personalization if the end-user has sufficient personalization rights.
 - 5 Click the **Differential build of project** button to rebuild your project.
 - 6 Log in to your Web application, browse to the updated form, and click the magnifying glass lookup icon () to display a pop-up lookup form.

The lookup field displays a list of values that match the Document Path you entered in step 3 above.
 - 7 If you want to change the field used for the lookup, click the **Personalize this page** link and select the new field you want to use.

Creating a Nested Document Lookup

To create a nested document lookup:

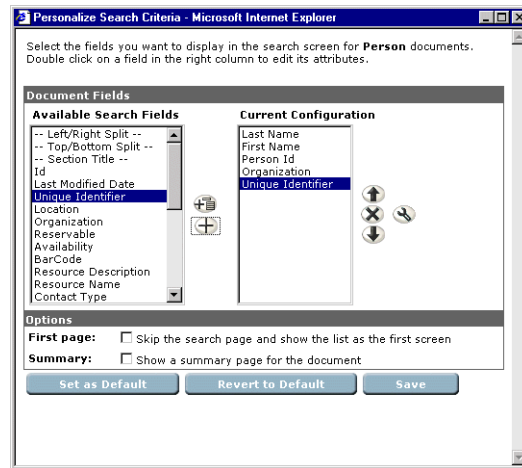
- 1 Right click the form to which you want to add the lookup.
- 2 Go to New > Field > Lookup.
- 3 Enter the following settings for the Data attribute:
 - Display Field—the name field that you want to be displayed in the Web application form when a user makes a selection from the lookup field. If you do not enter a value for this parameter it defaults to the Document Field parameter described below.
 - Document Field—the name of the key field used to uniquely identify each individual document record. The value of this field is used to lookup all other document fields of the subdocument. This value is posted to the onload script when a particular lookup entry is selected.
- 4 Enter settings for the following for the DocExplorer Adapter attributes:
 - Adapter—the name of the database adapter where your lookup information is stored.
 - Document Path—the name of the schema and subdocument name that defines the subdocument you want to lookup. The naming convention used here is schema name.subdocument name with a period (.) between them. For example, the entry employee.location will lookup the location subdocument from the employee schema.
- 5 Enter the following setting for the Link Parameters attribute:
 - Target Form. Enter docExplorer.documentlookup.start as the form name.
- 6 Click the **Differential build of project** button to rebuild your project.
- 7 Log in to your Web application, browse to the updated form and click the magnifying glass lookup icon () to display a pop-up lookup form.

The lookup field will display a list of values that match the Document Path you entered in step 3 above.
- 8 If you want to change the subdocument used for the lookup, click the **Personalize this page** link and select the new subdocument you want to use.

Using the Personalization Interface

You can personalize any Web application interface that displays a wrench icon in the top right of the interface frame. The wrench icon will appear only in activities where a Personalization form has been defined. The Personalization form determines what options are displayed in the Personalization pop-up window.

When you click on the Personalization icon, a pop-up window is displayed for the current form you are viewing.



All personalization pop-up windows have the format described below.

- **Available <columns>**—shows all the document fields and subdocument collections that can be added to the current form. The name of this column will vary depending on the type of form you are viewing. Studio generates the list of available fields by dynamically reading the schema used by the form. Any items listed between dashes are form components you can use to organize and arrange how document fields are displayed in the form.
- **Current Configuration**—shows all the document fields, subdocument collections, and displays components that have been selected for the current form. The first time a form is personalized, this column will be empty.

- Options—allows you to define how your Web application displays results and also determines whether users can update, create, or delete records from the back-end database system. Users will see the Options section only if they have sufficient Personalization rights.
- Set as Default—sets the revised configuration as the default.
- Revert to Default—removes all personalization entered by the end-user and returns the form to the default state. A default form may still display fields if the administrator or the form's schema has defined any default fields to be displayed.
- Save—saves and applies your Personalization changes to the current form.
- Close—closes the Personalization window saving any changes made to the form.

Note: The first time you browse to a form in a DocExplorer activity, the form will be blank. Administrators can add content to DocExplorer forms by personalizing each form or specifying what schema elements will appear in each form.

Adding Fields to a Form

To add fields to a form:

- 1 Select a field from the Available Columns list.
- 2 Click **Add**, and the field will appear in the Current Configuration list.
- 3 Click **Save**.

To arrange the order of fields:

- 1 Select a field from the Current Configuration list.
- 2 Click the up arrow or down arrow to change the field's position in the Current Configuration list.
- 3 Click **Save**.

Configuring Field Attributes

To configure field attributes:

- 1 Double-click a field from the Current Configuration list. A new Personalization pop-up window is displayed.
- 2 Enter the new field attributes:

- Label—the name to be used as the field label. This name appears next to the field in the interface.
 - Readonly—enter *true* if you do not want users updating information displayed in the field.
 - Required—enter *true* if this field must have a value before a form can be submitted.
- 3 Click Save.

Removing Fields from a Form

To remove fields from a form:

- 1 Select a field from the Current Configuration list.
- 2 Click the X button to remove the field.
- 3 Click Save.

6 Scripting

CHAPTER

Since scripts provide much of the functionality of Peregrine Web applications, this chapter provides an overview of how scripts are put together and used. You should be familiar with JavaScript or ECMAScript and should have access to the JavaDocs provided with your Web application installation.

Web applications built on the Peregrine OAA Platform use server scripts to query back-end databases and to format the results into XML documents based on schemas. Generally, you will only need to create new server scripts if you create new Web application forms. Most customized Web application forms do not require changes to the server script, but rather to the schema that the server script uses to display data. When you need to create or make changes to a server script, make sure you have created or activated a package extension in which to save your changes.

Tip: You can use the existing server scripts from your Web applications as templates for your custom server scripts. Try and find a server script that has similar functionality to what you want, and then copy and paste the server script into your Web application.

Types of Scripts

Peregrine Web applications use two types of scripting to transfer and format data between your back-end databases and Web application forms:

- **Server-side scripting**—Server-side scripts run from a Web server. Server-side scripts have access to both user-submitted form data and any data generated by a back-end system. The output of server-side scripts can be returned to both a back-end system and the remote browser. All Get-It Platform Web application server-side scripts are written in ECMAScript. An example of server-side scripting would be querying a back-end system for the list of items to be displayed in an order form.
- **Client-side scripting**—Client-side scripting runs from a JavaScript-capable browser. Client-side scripts have access to user data before it is submitted to a Web server and any back-end data that was uploaded with the current Web page. The output of client-side scripts can be used only by the client browser. All Peregrine OAA Platform client-side scripts are written in JavaScript. An example of using client-side scripting would be updating the total price displayed on an order form when an amount is entered in another field of the page.

Where Scripts are Stored

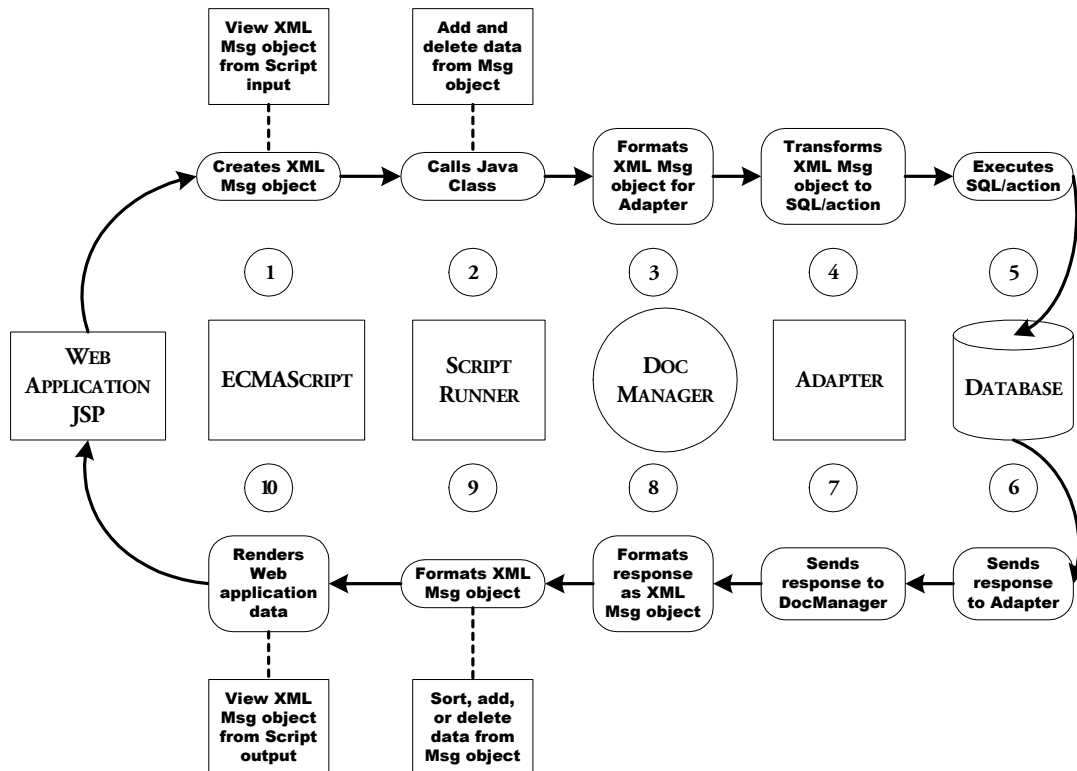
The following table describes how you can include both types of scripting into your projects.

Script type	Language used	Where created and stored
Server-side	ECMAScript	You can author server-side scripts only in Peregrine Studio. Each script then becomes an object available for use throughout the project.
Client-side	JavaScript	You can author client-side scripts outside of Studio and add them to a project Presentation folder. You can also include client-side scripts as part of the HTML code stored with a form. You must add your scripts to a Presentation folder of one of the Web application modules.

Studio stores all server-side ECMAScripts as part of your project file. At build time, Studio copies the scripts over into the deployment Web server's scripts folder and creates all necessary Web application JSP pages. At run time, the deployment Web server executes the Web application JSP pages along with any server-side scripts called by the JSP pages and sends the output to the client browser. The client browser will execute any client-side JavaScript present in the rendered JSP page.

How Scripts are Used

The Archway servlet supports several different methods to invoke and utilize scripts within Peregrine OAA Platform Web applications. The following sections describe the different ways in which ECMAScript and JavaScript can be used within your Web application.



Where Scripts are Used

Forms—Server Side

All Peregrine OAA Platform Web application forms support invoking onload server-side scripts. Typically, the onload script creates an XML message to gather and format information from a back-end database. The script message can contain queries or updates to the database or an XML document built from a schema. The scripts typically use a schema, one or more input parameters, and a back-end database query to create an XML document.

Many server onload scripts use one of the following Archway API calls:

- `sendDocQuery`—sends an SQL or XML document query to the back-end database. Archway queries only the tables and fields listed in the SQL statement or those listed in the schema definition and then return the results of the query as an XML document formatted as defined in the schema.
- `sendDocInsert`—sends an XML document to the back-end database that describes a new record. Archway creates the new record in the database using the table and field information supplied by the schema.
- `sendDocUpdate`—sends an XML document to the back-end database that describes an update to an existing database record. Archway updates the record using the table and field information supplied by the schema.
- `sendDocDelete`—sends an XML document to the back-end database that describes a record in the database to be deleted. Archway deletes the record using the table and field information supplied by the schema.

The Web applications typically use the following ECMAScript syntax to refer to schemas. For additional methods of formatting these messages, refer to the JavaDocs API documentation provided with your Web application installation.

```
archway.sendDocQuery( "adapter name", "schema name", input msg);  
archway.sendDocInsert( "adapter name", message object);  
archway.sendDocUpdate( "adapter name", message object);  
archway.sendDocDelete( "adapter name", message object);
```

For *adapter name*, enter the name for the back-end database adapter. For most applications, the adapter will be a two letter name (for example, *sc* for ServiceCenter).

For *schema name*, enter the name defined in the <document name="schema name"> element of the schema file.

For the *input msg*, enter the variable name that Archway uses to store input parameters for the ECMAScript function. The default input message is the *msg* object that Archway expects in all onload functions. The input message is the XML message containing the HTML page parameters.

For *message object*, enter a variable name that Archway can use to store the schema name and any input message.

The script sample below defines a variable called *msgReturn* that sends a document query to AssetCenter using the *empdetail* schema and any input parameters stored in the *msg* message object. The variable *msgReturn* then returns the result of the document query.

```
var msgReturn = archway.sendDocQuery( "ac", "empdetail", msg );
return msgReturn;
```

Client Side

The browser handles all client-side scripting when a user views a Web application.

Note: Peregrine does not provide customer support for custom client-side scripts.

Testing Scripts with URL Queries

You can test your server-side onload scripts and schemas by using URL queries to the Archway servlet.

Archway will invoke the server script or schema as an administrative user and return the output as an XML document. Your browser will need an XML renderer to display the output of the XML message.

Using URL queries can be useful for debugging your Web applications and for using the Archway servlet without requiring a Web application to process the data.

URL Script Queries Template

Archway URL script queries use the following format:

`http://server name/oaaservlet/archway?script name.function name`

- For *server name*, enter the name of the Java-enabled Web server. If you are testing the script from the computer running the Web server, you can use the variable `localhost` as the server name.

The servlet mapping assumes that you are using the default URL mapping that the Peregrine OAA Platform automatically defines for the Archway servlet. If you have defined another URL mapping, replace the servlet mapping with the appropriate mapping name.

- For *script name*, enter the name of the script as defined in Studio.
- For *function name*, enter the name of the function used by the script.

Note: If you are using Netscape 4.x, the browser will prompt you to save the XML output of the URL query to an external file.

URL Schema Queries Template

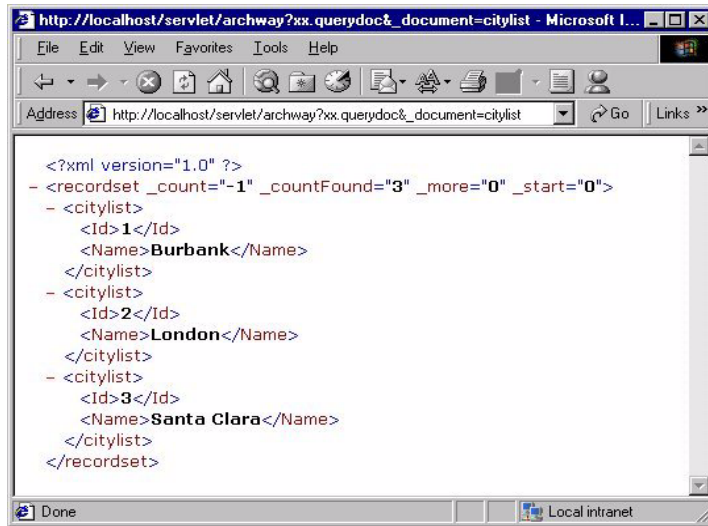
Archway URL schema queries use the following format:

`http://server name/oaaservlet/archway?adapter name.Querydoc&_document= schema name`

- For *adapter name*, enter the name for the back-end database adapter the schema uses. The adapter listed here will use the ODBC connection that you have defined in the Admin module Settings page. For most Peregrine OAA Platform Web applications, the adapter will be a two letter name.
- For *schema name*, enter the name defined in the `<document name="schema name">` element of the schema file.

The servlet mapping assumes that you are using the default URL mapping that is automatically defined for the Archway servlet. If you have defined another URL mapping, replace the servlet mapping with the appropriate mapping name.

Your script output should be similar to this.



```

<?xml version="1.0" ?>
- <recordset _count="-1" _countFound="3" _more="0" _start="0">
- <citylist>
  <Id>1</Id>
  <Name>Burbank</Name>
</citylist>
- <citylist>
  <Id>2</Id>
  <Name>London</Name>
</citylist>
- <citylist>
  <Id>3</Id>
  <Name>Santa Clara</Name>
</citylist>
</recordset>

```

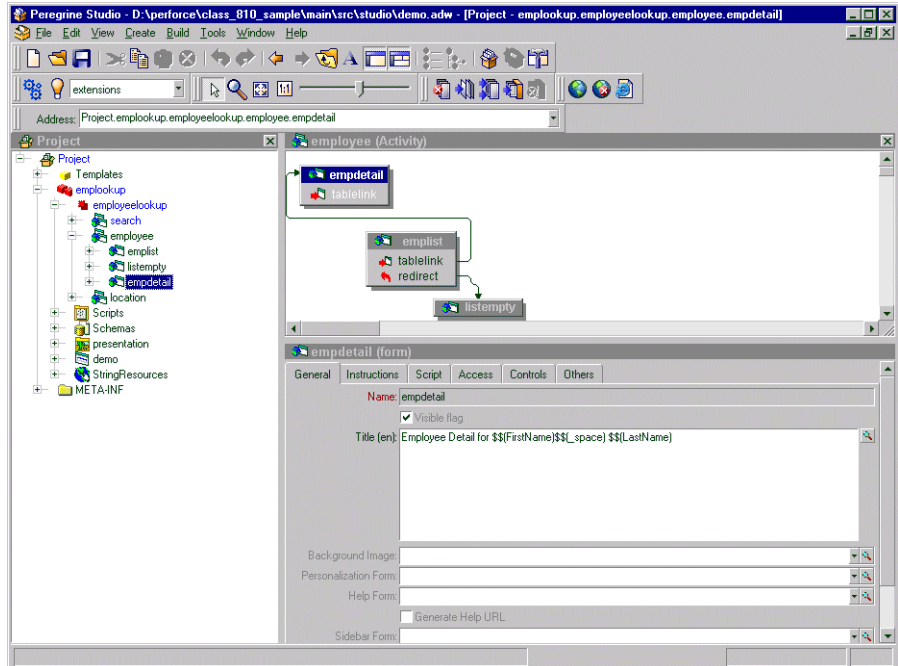
Note: If you are using Netscape 4.x, the browser will prompt you to save the XML output of the URL query to an external file.

Using Variables to Provide Script Data

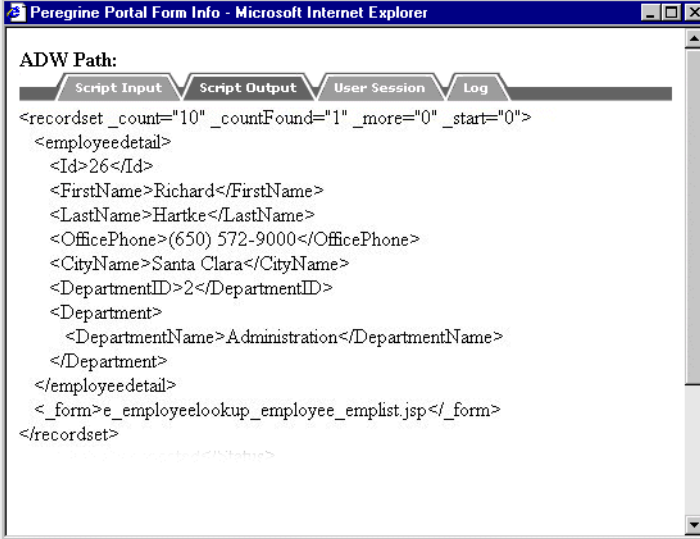
You can use variables to reuse the information gathered from your scripts in components such as form titles and instructions.

All variables begin with a double dollar sign and then display the variable name in parentheses; for example, `$$ (FirstName)`. All variable names map to an XML element name in the script output of a form. Thus the `$$ (FirstName)` variable maps to a `<FirstName>` element in the XML output of a script.

Studio maps the variables `$(FirstName)` and `$(LastName)` to XML elements `<FirstName>` and `<LastName>` in the script output.



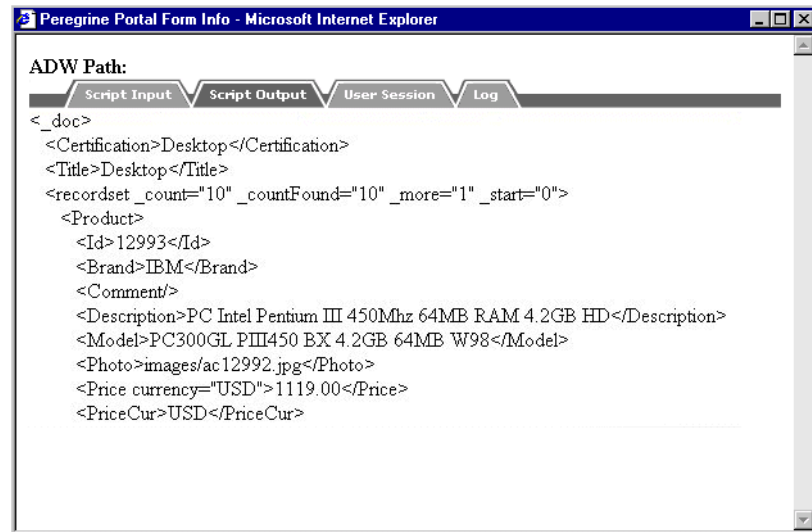
In the case of a search for an employee named Richard Hartke, the script output would look like the following.



```
ADW Path:
Script Input  Script Output  User Session  Log
<recordset _count="10" _countFound="1" _more="0" _start="0">
  <employeedetail>
    <Id>26</Id>
    <FirstName>Richard</FirstName>
    <LastName>Hartke</LastName>
    <OfficePhone>(650) 572-9000</OfficePhone>
    <CityName>Santa Clara</CityName>
    <DepartmentID>2</DepartmentID>
    <Department>
      <DepartmentName>Administration</DepartmentName>
    </Department>
  </employeedetail>
</recordset>
```

The contents of each variable are displayed in the form title.

Variable names can also include attribute names or nested elements names using a slash notation. For example, the Get-Resources buyer script uses the `$(Price/currency)` variable to pass information from the currency attribute of the `<Price>` element. Using the sample data, the `$(Price/currency)` variable would pass 1119.00 for the `<Price>` and USD for the currency.



```

ADW Path:
Script Input  Script Output  User Session  Log
<_doc>
<Certification>Desktop</Certification>
<Title>Desktop</Title>
<recordset _count="10" _countFound="10" _more="1" _start="0">
  <Product>
    <Id>12993</Id>
    <Brand>IBM</Brand>
    <Comment/>
    <Description>PC Intel Pentium III 450Mhz 64MB RAM 4.2GB HD</Description>
    <Model>PC300GL PIII450 BX 4.2GB 64MB W98</Model>
    <Photo>images/ac12992.jpg</Photo>
    <Price currency="USD">1119.00</Price>
    <PriceCur>USD</PriceCur>
  
```

Common Message Operations

The following section includes some common XML messages that can be used to modify server-side scripts.

- Creating a new generic message. `archway.sendDocQuery()` can process generic messages because the schema is passed as one of its other arguments.

```
var msgQuery = new Message();
```

- Creating a new message with a specific root element tag. `archway.sendDocUpdate()` and `archway.sendDocInsert()` use the root element tag as the schema to use for updating the database.

```
var msgRequest = new Message( "Request" );
```

- Adding an XML element to a message:

```
msgQuery.add( "LastName", "Jones" );
```

- Setting an XML element (will overwrite an existing element if one exists with the same name):

```
msgQuery.set( "LastName", "Jones" );
```

- Getting a value out of a message. This will return an empty string "" if the tag does not exist.

```
var strName = msg.get( "Name" );
```

- Getting a submessage out of a message:

```
var msgRequest = msg.getMessage( "Request" );
```

- Getting a list of submessages out of a message. This is useful for processing a response message with a list of records from sendDocQuery.

```
var list = msgResponse.getList( "Location" );
if ( list.getLength() == 0 )
msg.setCondition( "noresults" )
var i = 0;
while ( i < list.getLength() )
{
// process records in the list ...
}
```

- Logging the contents of a message. The output will go to archway.log if Debug Logging is enabled on the Settings Common tab. This should be done only during development—remove or comment out these lines on a production system because it is CPU-intensive on the server.

```
env.debuglog( "sendDocQuery returned the message " +
msgResponse.getContent() );
```

- Setting the response condition that is checked by redirections and access fields in different Studio elements:

```
if ( msg.get( "Name" ) == "" )
{ // need the name for the query
msgResponse.setCondition( "error" );
return msgResponse;
}
```

See the `com.peregrine.oaa.core.Message` JavaDoc for more information and examples.

About Script Pollers

The script poller process runs scripts in the background at a regular interval. The Archway servlet creates a script poller process when it finds a `scriptpollers.ini` file in one of the Files folders of your Studio project.

Peregrine OAA Platform Web applications use Script Pollers for the following types of tasks:

- Maintaining open connections to back-end databases.
- Clearing form data from caches.
- Checking for new messages or requests.

The scripts run by a script poller are written exactly like any other server-side ECMAScript: they are authored in Studio, include a header and functions, and must be stored in a Group of Scripts node. However, script poller scripts must have a function named *start* that runs once to initialize the script poller, and a function named *run* that runs once per time interval.

Each Web application can only have one `scriptpollers.ini` file, but each `scriptpollers.ini` file can contain multiple scripts. For each function that you want to run, you create a separate `scriptpollers.ini` file entry. The entry must contain the script name, function to be run, and the interval in seconds between each run time. You can set each function of a script to run at a different interval.

Enabling Script Polling

To enable script polling:

- 1 Log in to the Admin page (`admin.jsp`).
- 2 Click **Settings**.
- 3 On the Common tab, set Enable Script Pollers to *true*.

To create a group of files:

- 1 Open your Studio Project.
- 2 Select the Group of Modules node that contains the scripts you want to run from the script poller.
- 3 Click Create > Group of Files.

A node called Files will appear beneath the group of modules node you selected in step 2.

To add an entry to the scriptpollers.ini file:

- 1 Open your Studio project file.
- 2 Select the group of modules node that contains the scripts you want to run from the script poller.
- 3 Right click the Files node, go to New and click Ini File.
A new node called Ini File is created underneath the Files node.
- 4 Rename the new node to scriptpollers_ini.
- 5 In the OutputFileName field enter scriptpollers.ini.
- 6 Using the text editor window, add the entries listed below.

```
<pollers>
  <poller>
    <name>script.function</name>
    <interval>seconds</interval>
    <parms>script input parameters</parms>
  </poller>
</pollers>
```

- For *script.function*, enter the script and function name that you want the script poller to run.
 - For *seconds*, enter the number of seconds between each run of the script.
 - For *script input parameters*, enter any input parameters to send to the script.
- 7 If you want to have multiple scripts run with this script poller, add an additional <poller> entry for each script that you want to be run regularly.
 - 8 Save and build your project file.

The computer builds the scriptpoller.ini file to the WEB-INF\apps folder of your installation.

Stopping Script Polling

To stop script polling:

- 1 Open your Studio project.
- 2 Select the Group of Modules node that contains the scripts you want to stop.
- 3 Expand the Files node and then do one of the following:
 - Delete the scriptpollers_ ini file.
 - OR
 - Select the scriptpollers_ ini file and rename the OutFileName value from scriptpoller.ini to any other name. This will disable the script polling but still preserve the file in your WEB-INF\apps folder.
- 4 Save and build your project file.

The script poller is removed or renamed during the build.

Script Pollers in a Multiple JVM Environment

If you have installed multiple Java Virtual Machines (JVMs) on your server, you can designate the JVM that runs any particular script. If you do not designate a JVM, the script poller will behave as if it were in a single JVM environment.

To designate which JVM runs a script:

- 1 Open your Studio project file.
- 2 Select the Group of Modules node that contains the scripts you want to run from the script poller.
- 3 Expand the group of files node, and then select the scriptpoller_ ini file.
- 4 Using the text editor window, add the following entry between the <parms> tags for the script you want to change.

```
<Parms>
<ArchwayJVMName>Java Virtual Mahcine Name</ArchwayJVMName>
</Parms>
```

- 5 For *Java Virtual Machine Name* enter the name of the Java Virtual Machine that you want to run this script.

Sample Scripts

The following sections provide sample ECMAScripts and descriptions of how you can use them in your Web applications. The samples presented below use only server-side scripting.

It is beyond the scope of this guide to detail all that you can do with client-side scripting. However, a list of suggested reference materials for client-side scripting is provided on page 115.

General Script Samples

You can use ECMAScript to serve a number of different functions such as creating an XML document from a schema, running a SQL query, or formatting the data received from a database query. The following samples show some of the ways in which you can use ECMAScript in your Web applications.

Selecting a Field from a Schema

```
function getCityList ( msg )
{
//Query sample database for the records using the citylist
//schema
var msgQuery=newMessage();
msgQuery.set("_return", "Name");
var msgReturn=archway.sendDocQuery ("xx","citylist", msgQuery);

return msgReturn;
}
```

Input

A message object, msg. This script does not typically have input from any previous form. If you change this script to be part of a results form, then the input message could contain form fields or values for these form fields that could be added to the XML document produced.

Output

The script produces an XML document built from the schema and adapter specified in the sendDocQuery function. The XML output below is an example of the kind of data that could be returned using a similar script.

```
<recordset _count="-1" _countFound="3" _more="0" _start="0">
  <citylist>
    <Id>1</Id>
    <Name>Burbank</Name>
  </citylist>
  <citylist>
    <Id>2</Id>
    <Name>London</Name>
  </citylist>
  <citylist>
    <Id>3</Id>
    <Name>Santa Clara</Name>
  </citylist>
</recordset>
```

Although the `sendDocQuery` function specifies only the `Name` element, Archway automatically includes the `ID` element in the XML document produced. This is expected behavior of the Archway servlet.

Description

This script gathers a list of city names for the an employee search form. The `sendDocQuery` function uses a SQL-like query to select the `Name` element from a schema called `citylist`. You can use SQL statements in your script messages to limit or add to the elements returned by a schema. Likewise, you can use a SQL statement in place of a schema to specify exactly the data you want returned or to use complex selection criteria.

Calling Other Scripts and Combining the Results

```
function getSearchInfo( msg )
{
//Create empty variable msgResponse
var msgResponse = new Message();

//Call getDepList function and add results to msgResponse.
msgResponse.add( this.getDepList( msg ) );
// Call getCityList function and add results to msgResponse
msgResponse.add( this.getCityList( msg ) );

return msgResponse;
}
```

Input

A message object, msg. This script does not typically have input from any previous form. If you changed this script to be used as part of a results form, then the input message could contain form fields or values for these form fields that could be added to the XML document produced.

Output

The script produces an XML document built from two other scripts, getDepList and getCityList. Each script adds to the XML document stored in the msgResponse variable by running a sendDocQuery function with a schema. The XML output below is an example of the kind of data that could be returned using a similar script.

```
<_doc>
<recordset _count="-1" _countFound="19" _more="0" _start="0">
  <departmentlist>
    <Id>1</Id>
    <DepartmentName/>
  </departmentlist>
  <departmentlist>
    <Id>2</Id>
    <DepartmentName>Administration</DepartmentName>
  </departmentlist>
  <departmentlist>
```

```

        <Id>3</Id>
        <DepartmentName>Administrative Services</DepartmentName>
    </departmentlist>
<departmentlist>
    <Id>4</Id>
    <DepartmentName>Burbank Agency</DepartmentName>
</departmentlist>
...
</recordset>
<recordset _count="-1" _countFound="3" _more="0" _start="0">
    <citylist>
        <Id>1</Id>
        <Name>Burbank</Name>
    </citylist>
    <citylist>
        <Id>2</Id>
        <Name>London</Name>
    </citylist>
    <citylist>
        <Id>3</Id>
        <Name>Santa Clara</Name>
    </citylist>
</recordset>
<_form>e_employeelookup_search_search.jsp</_form>
</_doc>

```

Description

This script generates the city and department names that a user can select from in the employee search form. The .add function appends the output of the getDepList and getCityList functions to the msgResponse variable. The two script references use the relative naming convention (this) to indicate that the functions called are part of the same script as getSearchInfo.

Form Script Sample

Most ECMAScripts run during a form's onload processing. Typically, form scripts query and format data for display in a Web application form, but you can also use them to update existing database records or insert new ones. The following samples show server onload scripts used by a Web application that searches a database for employee information.

Creating an XML Document from a Schema

```
function getEmpList( msg )
{
//Add Department subdocument to the input message
var strReturn = msg.get("_return");
if ( strReturn.length > 0 )
    msg.set("_return", strReturn + ";Department");

//In msg, set sort to LastName and then FirstName
msg.add( "_sort", "LastName,FirstName" );

//Query sample database for the records using the
//employeeDetail schema and the criteria found in the msg object
var msgReturn = archway.sendDocQuery( "xx", "employeeDetail", msg );

//Test if the number of items returned is zero, if true set
//ListEmpty condition
if ( msgReturn.get("_countFound") == "0" )
    msgReturn.setCondition( "ListEmpty" );

//Return the contents of the msgReturn variable
return msgReturn;
}
```

Input

A message object, msg. This script has an input message from a previous search form. In this case, the input message is amended to include a subdocument, Department, in addition to any other input data passed to the script. This subdocument looks up the DepartmentName field data that the database stores in a separate table. In addition to adding a subdocument, the script sorts the input message by the LastName and FirstName elements. The following XML demonstrates what the input message would look like if a search were conducted on the CityName of Burbank (CityID=1).

```
<_doc>
<_form>e_employeelookup_employee_emplist.jsp</_form>
<_start>0</_start>
<_return>;employeedetail;CityName;OfficePhone;DepartmentName;FirstName;LastName;Id;</_return>
<_count>10</_count>
<_ctxobj/>
<_ctxidfld/>
<_ctxidval/>
<CityID>1</CityID>
<search>1</search>
<_blankFields>;FirstName;false;LastName;false;DepartmentID;false</_blankFields>
<_x>_y</_x>
<_callingform>e_employeelookup_search_search.jsp</_callingform>
<FirstName insertblank="false"/>
<LastName insertblank="false"/>
<DepartmentID insertblank="false"/>
</_doc>
```

Output

The script produces an XML document built from the schema and adapter specified in the sendDocQuery function. The XML output below is an example of the kind of data that could be returned using a similar script.

```
<recordset _count="10" _countFound="2" _more="0" _start="0">
  <employeedetail>
    <Id>10</Id>
    <FirstName/>
    <LastName>Burbank Agency</LastName>
    <OfficePhone>(408) 422-5501</OfficePhone>
```

```

    <CityName>Burbank</CityName>
    <DepartmentID>16</DepartmentID>
    <Department>
      <DepartmentName>Sales</DepartmentName>
    </Department>
  </employeeedetail>
<employeeedetail>
  <Id>11</Id>
  <FirstName/>
  <LastName>Burbank Unit</LastName>
  <OfficePhone>(650) 572-9000</OfficePhone>
  <CityName>Burbank</CityName>
  <DepartmentID>19</DepartmentID>
  <Department>
    <DepartmentName>Technical Support</DepartmentName>
  </Department>
</employeeedetail>
<_form>e_employeelookup_employee_emplist.jsp</_form>
</recordset>

```

Description

This script displays the results list generated by the search form. The script uses two functions to change the data in the msg input message object. The first function checks the input message to determine the number of elements returned by the search results. If there any search results to return, the scripts adds the Department subdocument to the msg message object. The second function sorts the input message by LastName and then FirstName. Using the adapter name and document schema name, this script then runs a SendDocQuery function to gather any search results that match those listed in the input message. The script then checks the <_countfound> tag generated by the query and determines if the return list is empty. If the list is empty, the script sets the msgReturn variable to the ListEmpty condition. This condition redirects users to the listempty form.

Script Poller Sample

Scripts added to the script poller are the same as all other ECMAScripts and may use any of the scripting techniques described above.

Get-It Platform Web applications use script pollers to keep the connection alive between the application and the back-end systems, ServiceCenter and AssetCenter.

The `scriptpoller.ini` file defines the interval between runs and the input parameters used.

Note: Get-It Platform Web applications include adapters for AssetCenter and ServiceCenter that automatically reconnect when a connection is lost. The following script poller scripts are provided for illustration purposes only.

Maintaining a Connection to AssetCenter

```
//-----
// KeepAliveAC
// (c) Peregrine Systems 2000
// Creation: August 2000
//
// Used to poll and keep-active the AssetCenter connection by
// performing a query at regular intervals. If the query fails,
// the archway connection will then be disconnected and
// re-connected.
//
// Two functions are defined:
// start() - executes exactly once
// run() - executes on the polling interval
//
//-----

import global;

//-----
// Start function .. can build parameters for run method
//-----
```



```

function start( msg )
{
  // Before trying to run this script on a regular basis, make sure
  // the ac adapter is registered

  if ( archway.getArchway().supportsAdapter( "ac", false ) )
  {
    var msgRet = new Message();
    msgRet.set("message","ok");
    return msgRet;
  }
  else
  {
    env.log( "AC Adapter is not registered; terminating" );

    return null; // Suppress calling of run function
  }
}

//-----
// run function .. subsequent invocations
//-----
function run( msg )
{
  var msgRes = null;
  var msgRet = new Message();

  msgRes = archway.sendQuery( "ac",
    "SELECT IOptId, memOptValue FROM amOption WHERE OptSection='TimeZone'
    AND OptEntry='DBSERVER_TZ'",
    0, 1 );

  if ( msgRes.get( "message" ) ) // if nothing is found, try to reconnect
  {
    env.debuglog( "Attempting to reconnect to AssetCenter." );
    var adapter = Archway.Archway.getInstance().findAdapter( "ac" );
    if ( adapter != null )
    {
      adapter.disconnect();
    }
  }
}

```

```
        adapter.connect();
    }
}
msgRet.set("message","ok");

return msgRet;
}
```

Input

A message object, msg, although the script does not use the input message.

Output

The start function of the script checks to see if there is an AssetCenter adapter. If there is an adapter present, the script returns the XML document `<message>ok</message>`. If the adapter is not present, the message document will be blank and the script will return a null message to stop running.

The run function of the script runs a `sendDocQuery` to AssetCenter. If the query returns the blank XML document `<message></message>`, then the script will disconnect and reconnect the AssetCenter adapter. If the query returns a response, then the connection is successful, and the script returns the XML document `<message>ok</message>`.

Description

This script sends a periodic query to AssetCenter to see if it responds. If AssetCenter returns a blank message, then the script will disconnect and reconnect the adapter. If AssetCenter responds with a message, then the connection is alive and the script returns an OK message.

References

This section contains reference material to help you with your scripting.

Sources for Client-side JavaScript

- Devguru (JavaScript, VB script, HTML, etc.): <http://www.devguru.com/>
- HTML Writer's Guild: <http://www.hwg.org/>
- *JavaScript, The Definitive Guide*, David Flanagan, 3rd Edition, O'Reilly Publishing.
- JavaScript articles at IRT.org: <http://www.tech.irt.org/articles/script.htm>
- JavaScript Made Easy: <http://www.easyjavascript.com/>
- JavaScript Source: <http://javascriptsource.com/>
- JavaScript Source master list: <http://javascript.internet.com/master-list/>
- Netscape's Developer Site: <http://developer.netscape.com>
- Netscape's online JavaScript documentation.:
<http://developer.netscape.com/docs/manuals/index.html?content=javascript.html>
- Web Monkey: <http://www.webmonkey.com/>
- ZDNet JavaScript introduction:
<http://www.zdnet.com/devhead/filters/0,,2133214,00.html>

JavaDocs for the Main Archway Package

For in-depth information about the Archway servlet and all the functions it supports, refer to the JavaDocs that are installed with your application. The JavaDocs are located in the \docs\api folder of your installation. To view the docs, launch the index.html file from this folder.

7 Document Schema Definitions

CHAPTER

A document schema definition (also called a schema) is an XML file that instructs the Document Manager how to query back-end databases and generate XML documents containing the query response. Schemas are mapping tools that determine which XML tags used in the forms map to the table and field names in a given back-end database. These generated XML documents provide the data that your Web applications display and process.

All document schemas consist of two types of definitions:

- Base definitions—a definition of the XML tags to be generated. The Document Manager will create the XML tags listed in the schema when it performs a document query. The *name* of the XML tag generated is determined by the attribute *name* schema entry. The *type* of information stored in the XML tag is determined by the attribute *type* schema entry. The schema entries that define XML tags are collectively referred to as the schema base definitions.

- **Derived definitions**—a definition of the tables and fields each XML tag will map to in the back-end database. Archway will query the tables and fields listed in the schema and add the results of the query to the appropriate XML tags. The name of the table queried is determined by the document table schema entry. The name of the field queried is determined by the attribute field schema entry. The schema entries that define the database mappings are collectively referred to as the schema derived definitions.

Note: The document schema definitions used by Studio are not the same as the schemas being proposed and developed by the W3C. Studio schemas are used to map logical document and field names in Studio to back-end database table and field names.

The Studio interface requires that schemas be grouped under a *group of schemas* node. A group of schemas can be added only to a *group of modules*.

You will need to create schemas to instruct the Document Manager how to query, update, or insert information to your back-end databases. Your schemas determine what XML documents the Archway servlet generates, which in turn determine what data your Web applications can display and process. Generally, you will need to create a schema for each back-end database table you want to query, update, or insert. You can however, create one schema that maps one set of interface fields to several different back-end databases. When you need to create or make changes to a schema, make sure you have created or activated a package extension in which to save your changes.

Tip: You can use the existing schemas from your Web application as templates for your custom schemas. Try and find a schema that has similar functionality to what you want, and then copy and paste the schema into your Web application.

How Schemas are Used

Each of the document definition types, Base or Derived, has its own set of schema entries and requirements.

Your Web applications can use a schema in one of two ways:

- A server ECMAScript can use a schema to construct a message. There are four API calls that use schemas to interpret messages: `sendDocQuery`, `sendDocInsert`, `sendDocUpdate`, and `sendDocDelete`.
- A form component, such as a simple table or a selectbox, can use a schema to determine what information is to be displayed by the component. Typically, the form components use the record attribute to determine what part of a message to display.

Schemas with ECMAScript

All Peregrine OAA Platform Web application forms support invoking onload server-side scripts. Typically, the onload script will be used to create an XML message to gather and format information from a back-end database. The script message can contain hard-coded SQL statements to the database or an XML document built from a schema. The scripts typically use a schema to create an XML document constructed from one or more input parameters and any queries to the back-end databases.

Most server onload scripts will use one of the following Archway API calls:

- `sendDocQuery`—sends a SQL or XML document query to the back-end database. OAA will only query the tables and fields listed in the SQL statement or those listed in the schema definition and then return the results of the query as an XML document formatted as defined in the schema.
- `sendDocInsert`—sends an XML document to the back-end database that describes a new record. Archway will create the new record in the database using the table and field information supplied by the schema.
- `sendDocUpdate`—sends an XML document to the back-end database that describes an update to an existing database record. Archway will update the record using the table and field information supplied by the schema.
- `sendDocDelete`—sends an XML document to the back-end database that describes a record in the database to be deleted. Archway will delete the record using the table and field information supplied by the schema.

ECMAScript Syntax

Peregrine OAA Platform Web applications use the following ECMAScript syntax to refer to schemas. For additional methods of formatting these messages, refer to the OAA API documentation provided with your installation.

```
archway.sendDocQuery( "adapter name", "schema name", input msg);  
archway.sendDocInsert( "adapter name", message object);  
archway.sendDocUpdate( "adapter name", message object);  
archway.sendDocDelete( "adapter name", message object);
```

- For *adapter name*, enter the name for the back-end database adapter. The adapter listed here will use the ODBC connection that you have defined in the achway.ini file. For most applications, the adapter will be a two letter name.
- For *schema name*, enter the name defined in the <document name="schema name"> element of the schema file.
- For the *input msg*, enter the variable name of a message that OAA uses to store input parameters for the ECMAScript function. The default input message is the msg object that is defined in all onload functions. The input message is the XML message containing the HTML page parameters.
- For *message object*, enter a variable name of a message object containing a schema name and any input parameters.

For example, the script sample below will define a message called msgReturn that contains the results of a document query to AssetCenter using the empdetail schema and any input parameters stored in the msg message object. The result of the document query is then returned as the variable output.

```
var msgReturn = archway.sendDocQuery( "ac", "empdetail", msg );  
return msgReturn;
```


Identifying the Back-end System Version

Document schemas can be modified to allow mapping to specific versions of the back-end systems. Each adapter determines the version of the back-end system to which it is connected.

The extension is made possible by an optional “version” attribute that can be used as shown in the following example for AssetCenter:

```
<schema>
  <documents name="base">
    .... (base definition)
  </documents>

  <documents name="ac" version="3.02">
    .... (AC 3.02 definition)
  </documents>

  <documents name="ac" version="3.5">
    .... (AC 3.5 definition)
  </documents>
</schema>
```

The version of the back-end system is determined at run time using an Adapter API.

The following version rules apply when searching for a schema mapping:

- If a mapping has no version: automatically accepted as default.
- If mapping version > backend Version: mapping is rejected.
- If a mapping version <= backend version: mapping is accepted, but the DocManager keeps looking for the highest possible version.

In summary, if a schema and adapter have version information, the highest possible mapping version that is no higher than the current adapter version is used.

AssetCenter Feature Links

AssetCenter feature links can be used in schemas for read or write access in the same way as any other field or link from AssetCenter.

As an example of how this can be done, consider the following example of the `fv_ShipToContact` feature that implements a link to the employee table in the purchase order table in the Web application, `Get-Resources`. (See `GRPurchaseOrder` schema, `Project.resources.Schemas.GRPurchaseOrder`).

```
<attribute name="ShipToContactId" field="fv_ShipToContact" access="uiq"/>
<attribute name="RShipToContactId" field="fv_ShipToContact.ImplDeptId"
access="r"/>
<attribute name="ShipToContactDesc" field="fv_ShipToContact" access="r"/>
```

- The first line is used to set (update or insert) and search against a specific contact ID. It is a numeric field.
- The second line is used to read the purchase order's contact ID. It is also a numeric field.
- The third line returns the PO's contact description. This is a text field.

Compare this to how `ShipToContact` would be set up as a regular link in AssetCenter:

```
<attribute name="ShipToContactId" field="lShipToCntcId"/>
<attribute name="ShipToContactDesc" field="ShipToContact" access="r"/>
```

There is only one attribute that gives read/write/search access to the contact ID (numeric). It maps to the `lShipToCntcId` field in the `amPOrder` table. This field is the foreign key representing the link in the `amPOrder` table. In the implementation using a link feature, there is no access to a foreign key; therefore two attributes must be used.

The other attribute `ShipToContactDesc`, does not differ in its implementation, whether you are using a link or a link feature.

Schema Naming Conventions

Each schema you create should have a unique name to prevent data errors. The schema name should meet the following criteria:

- Unique from any other schema name in the Studio project.
- Unique from any field name within the schema.

Schema Elements And Attributes

All schemas use a standard set of XML elements and attributes that the Document Manager recognizes. The following sections describe the XML elements and associated attributes that you can use to create valid schemas for your Web application.

<schema>

The <schema> element is a required container for all schema definitions. The only valid child element is the <documents> element. The <schema> element does not have any attributes.

<documents>

The <documents> element encloses the two types of document definitions: base document definitions and derived document definitions. The only valid child element of the <documents> element is the <document> element. All schemas are required to have at least two <documents> elements. One <documents> element provides the base derivations where the XML tags are defined, and the second and later <documents> elements define the back-end database derivations.

The <documents> element can have two attributes: name and version.

- name—defines what document derivation the child elements describe. All schemas must have a <documents> element where the name attribute is defined as base. This <documents> element will contain the definition of the XML tags to be generated by the Archway document message. The second and later <documents> elements will have the document derivations for your database tables and fields. The name of the database derivations <documents> element is set to the adapter name used to connect to the back-end database. For example, the following schema entries create AssetCenter document derivations using the name ac and create ServiceCenter document derivations using the name sc.

```
<documents name="base">
  <!--XML tag definitions-->
  ...
</documents>

<documents name="ac">
  <!--AssetCenter derivations-->
  ...
</documents>

<documents name="sc">
  <!--ServiceCenter derivations-->
  ...
</documents>
```

- version—defines the version of the back-end database to which the derivation mappings apply. You can use the version attribute to define multiple versions of database derivations. In this scenario each database version would have its own <documents> element with the specific database derivations that apply to that version of the database. If you include a version attribute in a schema, the Archway servlet will query the database for its version number and then use the matching database derivations for that version. For example, the following schema entries provide document derivations for different versions of AssetCenter.

```
<documents name="ac" version="3.02">
  <!--AssetCenter 3.02 derivations-->
  ...
</documents>
```

```
<documents name="ac" version="3.5"
  <!--AssetCenter 3.5 derivations-->
  ...
</documents>
```

<document>

The <document> element defines the top level tag in the XML document created by an Archway message. The only valid child elements of the <document> element are <attribute>, <collection>, and <document>.

A <document> element can be nested inside other <document> elements in one of two ways:

- You can define the nested <document> element in-place.
- You can define the nested <document> element by reference to another schema file or later in the schema.

<document> attributes

The <document> element can have up to five attributes: name, table, field, joinfield, joinvalue, and joinvalue.

name

The name attribute is a mandatory attribute that uniquely identifies the XML document generated by Archway. The name you enter for this attribute must match the schema name and cannot contain spaces. For example, if you have a schema called `departmentlist.xml`, then the attribute name must be `departmentlist`. You must use the same name attribute for the <document> element of both the base document definitions and the derived document definitions.

table

The table attribute is a mandatory attribute that identifies the source table for the derived document definitions. Each derived document definition can only have one table attribute. At a minimum, at least one attribute, the primary key (defined as the Id attribute) must be located in this table. To call data from other tables you can either nest document elements or use table link references.

field

The field attribute is a mandatory attribute that identifies the source field for the derived document definitions. This attribute can be used for both document queries and document updates or insertions. Each field attribute maps to one particular field or link in the database table. To call a field from a linked table, you will need to create a nested document entry to define the linked table and then include a separate <attribute> element to call the field.

joinfield

The joinfield attribute is an attribute used to query information from linked database tables. The joinfield attribute is used in conjunction with the table, field, and joinvalue attributes to query documents from linked database tables (a linked table contains fields that are lookup values to other database tables). The joinfield defines what field will be the selection criteria in a SQL WHERE clause. The SQL query equivalent of the joinfield is:

```
SELECT <Field> FROM <Table containing lookup information> WHERE
<joinfield>=<joinvalue>
```

If no joinfield is defined, then Archway will use the Id field name defined in the parent table as the joinfield. The parent table is defined by the table attribute in the first instance of the document element.

joinvalue

The joinvalue attribute is an attribute used to query information from linked database tables. The joinvalue attribute is used in conjunction with the table, field, and joinfield attributes to query documents from linked database tables (a linked table contains fields that are lookup values to other database tables). The joinvalue defines what value a field must have in a SQL WHERE clause. The SQL query equivalent of the joinvalue is:

```
SELECT <Field> FROM <Table containing lookup information> WHERE
<joinfield>=<joinvalue>
```

If no joinvalue is defined, then Archway will use the name attribute as the joinvalue.

Note: The joinvalue can either be a field from the back-end database or can be an <attribute> defined elsewhere in the schema.

<attribute>

The <attribute> element defines the XML tags that are created in an Archway message, and also defines which back-end database fields provide the content of the XML tags. The attributes listed under the base document definitions define the XML tags to be generated in an Archway message. The attributes listed under the derived document definitions define the back-end database fields where document queries, updates, or insertions are to be made. The only valid child elements of the <attribute> element are <collection> and <document>.

The <attribute> element uses only child elements to provide link information for fields in lookup tables.

name

The name attribute is a mandatory attribute that defines the XML tag to be generated in an Archway message. Whatever name you enter for this attribute will be used as the tag name in the Archway message. For example, if you define an <attribute> element with the name ="Price", then when Archway creates a message using this schema you should see a <Price> tag created in the message output. The value you enter for this attribute is case sensitive and can only contain alphanumeric characters.

All schemas must contain at least one <attribute> element where the name attribute is set to Id. The <attribute name="Id"> element is required to uniquely identify each record in an Archway-generated XML document.

type

The type attribute is an optional attribute that defines the type of data that will be stored in the XML tag. This attribute is used with Personalization to determine what type of form component will be rendered for a given element. Archway will not perform any data type validation for the contents of the XML tags. The type attribute can only be defined in the base document definitions section of the schema.

Type values

The following values are supported for this attribute:

- attachment—data contained in the XML tag will be a file name and path to an attachment. Elements of this type will be rendered as an attachment control in personalized forms.
- boolean—data contained in the XML tag will be a true or false string. Elements of this type will be rendered as a check box in personalized forms.
- date—data contained in the XML tag will be date. Archway will not localize the date format. Elements of this type will be rendered as a date edit control that includes a popup calendar in personalized forms.
- datetime—data contained in the XML tag will be a combined date and time listing. Archway will not localize the date or time format. Elements of this type will be rendered as a time edit control in personalized forms.
- id—data contained in the XML tag will be a number that uniquely describes a back-end database record.
- image—data contained in the XML tag will be an image. Elements of this type will be rendered as an imagefield in personalized forms.
- link—data contained in the XML tag will be a link to a subdocument elsewhere in the schema. Elements of this type will be rendered as a lookup field in personalized forms.
- memo—data contained in the XML tag will be a text string. Elements of this type will be rendered as a multi-line edit box in personalized forms.
- money—data contained in the XML tag will be a currency amount. Elements of this type will be rendered as a money field that includes a currency selection tool in personalized forms.
- number—data contained in the XML tag will be an integer. Elements of this type will be rendered as an editfield with spinner buttons in personalized forms.
- preload—data contained in the XML tag is an executable script.
- string—data contained in the XML tag is text. Elements of this type will be rendered as an editfield in personalized forms.

- time—data contained in the XML tag is a time listing. Archway will not localize the time format. Elements of this type will be rendered as a time edit control in personalized forms.
- url—data contained in the XML tag is a Web site address. Elements of this type will be rendered as an HREF link icon in personalized forms.

shortdesc

The shortdesc attribute is an optional attribute used with Personalization. This attribute defines what label should be used to describe the attribute in a personalized form. The contents of this attribute can be any text string.

search

The search attribute is an optional attribute used with Personalization to define the default content of a personalization form. When this attribute is set to true, the element will be displayed in the personalization search form.

list

The list attribute is an optional attribute used with Personalization to define the default content of a personalization form. When this attribute is set to true, the element will be displayed in the personalization list form.

detail

The detail attribute is an optional attribute used with Personalization to define the default content of a personalization form. When this attribute is set to true, the element will be displayed in the personalization detail form.

create

The create attribute is an optional attribute used with Personalization to define the default content of a personalization form. When this attribute is set to true, the element will be displayed in the personalization create form.

field

The `field` attribute is a mandatory attribute that defines the target field in the back-end database to be used by Archway document queries. The field attribute must match the field name in the back-end database. The field attribute can only be defined in the derived document definitions section of the schema.

link

The `link` attribute is a required attribute when you are attempting to update or insert information into linked database tables. The link attribute is used in conjunction with the `linktable`, `linkfield`, `linktype`, and `linkkey` attributes. The link attribute defines the database field in the parent table containing a lookup or link value to a linked table. This attribute is also the target database field where Archway will update or insert information. The link attribute can only be defined in the derived document definitions section of the schema.

linktable

The `linktable` attribute is a required attribute when you are attempting to update or insert information into linked database tables. The linktable attribute is used in conjunction with the `link`, `linkfield`, `linktype`, and `linkkey` attributes. The linktable attribute defines the database linked table where Archway will find the source information for updates or insertions. The linktable attribute can only be defined in the derived document definitions section of the schema.

linkfield

The `linkfield` attribute is a required attribute when you are attempting to update or insert information into linked database tables. The linkfield attribute is used in conjunction with the `link`, `linktable`, `linktype`, and `linkkey` attributes. The linkfield attribute defines the database source field where Archway will find update or insert information. The linkfield attribute can only be defined in the derived document definitions section of the schema.

linktype

The linktype attribute is an optional attribute when you are attempting to update or insert information into linked database tables. The linktype attribute is used in conjunction with the link, linktable, linkfield, and linkkey attributes. The linktype attribute defines how Archway will perform a document insert or update.

The linktype can be set to one of two values.

- Soft—Archway will query the database using the locations defined in linktable and linkfield, and set the link attribute to the query result.
- Hard—Archway will create a new record in the database in the location defined by the linktable and linkfield attributes. The linkkey value will be retrieved for the new record and saved in the link attribute.

If the linktype is not defined, the linktype will default to the soft value. The linktype attribute can only be defined in the derived document definitions section of the schema.

linkkey

The linkkey attribute is an optional attribute when you are attempting to update or insert information into linked database tables. The linkkey attribute is used in conjunction with the link, linktable, linkfield, and linktype attributes. The linkkey attribute defines what field will be the selection criteria in a SQL WHERE clause. The SQL query equivalent of the linkkey is:

```
SELECT <linkfield> FROM <linktable> WHERE <linkkey>=<the value specified in the update or insert message>
```

The results of the query are stored in the field defined by the link attribute.

If no linkkey is defined, then Archway will use the link attribute as the linkkey. The linkkey attribute can only be defined in the derived document definitions section of the schema.

How to Create Schemas

The Studio interface requires that schemas be grouped under a group of schemas node. You can only add a group of schemas to a Group of Modules.

Creating Groups of Schemas

To create a group of schemas:

- 1 Right-click the module to which you want to add a group of schemas.
- 2 Point to New, and then click Group of Schemas. A new node will appear with the name *Schemas*.

Creating Schemas

To create a schema:

- 1 Right-click the group of schemas node to which you want to add a schema.
- 2 Point to New, and then click Raw Schema. A new node appears with the name *Schema*. Studio displays the content of your schema in a text editor window. Use the text editor window to review and edit the XML source code of the schema.

Schema Template

Your schema must follow this template.

```
<?xml version="1.0"?>
<schema>

<!--
=====
Base Definitions: XML tags and data types defined
=====
-->

<documents name="base">

<document name="schema name">
  <attribute name="XML tag name" type="data type"/>
  ...
</document>
```

```

</documents>

<!--
=====
Database Definitions: Mappings from XML tags to database tables
and fields.
=====
-->

<documents name="adapter name">
  <document name="schema name" table="table name">
    <attribute name="XML tag name" field="field name"/>
    ...
  </document>
</documents>

</schema>

```

Schema Template Entry Descriptions

This table describes the schema entries from the schema template.

Schema Entry	Description
<?xml version="1.0"?>	Since all schemas are written as XML files, they must begin with the XML definition tag.
<schema>	This is the container element for all schema entries. All schema definitions must be enclosed between this tag and the final closing schema tag.
<documents name="base">	This is the container element for the Base Document Definitions. The child <i>document</i> and <i>attribute</i> elements of this element will define what XML tags are generated by an Archway message. The <i>name</i> attribute for this element must always be <i>base</i> .
<document name="schema name">	The first instance of this element defines the schema name. The name attribute must match the schema file name. For example, if you create a schema called locationdetail.xml, you must enter locationdetail as the name attribute.
<attribute name="XML tag name" ...	This element defines what XML tag will be generated by the Archway message. The name of the XML tag will be the name you enter in quotes for the name attribute.

Schema Entry	Description
<code><attribute ... type="data type"/></code>	This element defines what type of information will be stored in the XML tag. Valid information types include: Id, string, number, date, and url.
<code><documents name="adapter name"></code>	This is the container element for the Derived Document Definitions. The child document and attribute elements of this element will define what back-end database tables and fields are queried or updated by an Archway message. The name attribute for this element defines what adapter will be used for the Archway message. The adapter name must match an adapter defined on the Settings page of the Admin module.
<code><document ... table="table name"></code>	This element defines the database table to be used by the Archway message. The table attribute for this element defines the back-end database table to be queried or updated.
<code><attribute ... field="field name"/></code>	This element defines the database field to be used by the Archway message. The field attribute of this element defines the back-end database field to be queried or updated.

Using Nested <Document> Elements to Call Linked Tables

Consider a sample Employee Lookup Web application that uses a schema called `employeedetail` to query the sample database for employee contact information. The sample database stores the needed employee contact information in two linked tables:

- The Employee table stores information about the employee and has a lookup field to the Department table.
- The Department table stores information about the company's departments.

Examples

The following examples illustrate how to query the fields stored in the tables described above. While each example uses a different document nesting method, all three will accomplish the same tasks:

- Select a particular database record from the Employee table based on the script output of the previous form. Search the Employee table and select the record whose ID field matches the script input.
- Select the `Department_ID` field for the record in item 1 (the `Department_ID` field is a lookup field to the Department table).

- Search the Department table and select the record whose DepartmentID field matches the value in item 2.
- Select the DepartmentName field for the record selected in item 3. Return the value of DepartmentName field as the contents of the DepartmentName XML tag.

Nesting the <Document> Element In-place

When you define all table and field attributes of a nested <document> element within the parent <document> element, you are nesting the element in-place. This method stores all schemas in one schema file and presents the schema mappings in a linear flow.

Note: If you use this method of nesting you will not be able to use this schema with any Personalization features. To use Personalization, you must use one of the two nesting-by-reference methods described later in this chapter.

```
<!--Begin Base Document Definitions-->
<documents name="base">

  <!--Begin employeedetail document-->
  <document name="employeedetail">
    <attribute name="Id" type="id"/>

    <!--Begin nested Department document-->
    <document name="Department">
      <attribute name="DepartmentName" type="string"/>
    </document>
    <!--End nested Department document-->

  <!--End employeedetail document-->
  </document>

<!--End Base Document Definitions-->
</documents>

<!--Begin Derived Document Definitions-->
<documents name="xx">

  <!--Begin employeedetail document-->
```

```

<document name="employeedetail" table="Employee">
  <attribute name="Id" field="ID"/>

  <!--Begin nested Department document-->
  <document name="Department" field="Department_ID"
    table="Department" joinfield="ID" joinvalue="DepartmentID">
    <attribute name="DepartmentName" field="DepartmentName"/>
  </document>
  <!--End nested Department document-->

  <!--End employeedetail document-->
</document>

<!--End Derived Document Definitions-->
</documents>

```

XML Output

The Archway Document Manager produces XML output with the following structure. The content of the XML elements varies depending on the actual user record selected.

```

<employeedetail>
  <Id>17156</Id>
  <Department>
    <DepartmentName>Administration</DepartmentName>
  </Department>
</employeedetail>

```

Nesting the <Document> Element by Reference to <Collection>

You can reference any <document> element in your schema in a separate section by using the <collection> element. This method still stores all schemas in one schema file, but the schema mapping for each document have their own dedicated section.

```

<!--Begin Base Document Definitions-->
<documents name="base">

  <!--Begin employeedetail document-->
  <document name="employeedetail">

```



```

    <attribute name="Id"         type="id"/>

    <!--Begin reference to Department document-->
    <collection name="Departments">
      <document name="Department"/>
    </collection>
    <!--End reference to Department document-->

  <!--End employeedetail document-->
</document>

  <!--Begin nested Department document-->
  <document name="Department">
    <attribute name="DepartmentName" type="string"/>
  </document>
  <!--End nested Department document-->

<!--End Base Document Definitions-->
</documents>

<!--Begin Derived Document Definitions-->
<documents name="xx">

  <!--Begin employeedetail document-->
  <document name="employeedetail" table="Employee">
    <attribute name="Id"         field="ID"/>
    <!--Begin reference to Department document-->
    <collection name="Departments">
      <document name="Department"/>
    </collection>
    <!--End reference to Department document-->

    <!--End employeedetail document-->
  </document>

  <!--Begin nested Department document-->
  <document name="Department" field="Department_ID"
    table="Department" joinfield="ID" joinvalue="DepartmentID">
    <attribute name="DepartmentName" field="DepartmentName"/>
  <!--End nested Department document-->

```

```
</document>
```

```
<!--End Derived Document Defintions-->
```

```
</documents>
```

XML Output

The Archway Document Manager will produce XML output with the following structure. The content of the XML elements will vary depending on the actual user record selected.

```
<employeedetail>
```

```
<Id>17156</Id>
```

```
<Departments>
```

```
<Department>
```

```
<DepartmentName>Administration</DepartmentName>
```

```
</Department>
```

```
</Departments>
```

```
</employeedetail>
```

Nesting the <Document> Element by Reference Docname

The final nesting method uses the *docname* attribute to reference other <document> elements that are defined in separate schema files. Each nested <document> element will reference its own schema file.

```

<!--Begin Base Document Definitions-->
<documents name="base">

  <!--Begin employeedetail document-->
  <document name="employeedetail">
    <attribute name="Id"      type="id"/>

    <!--Begin reference to Department document-->
    <document name="Departments"  docname="departments"/>
    <!--End reference to Department document-->

  <!--End employeedetail document-->
  </document>

<!--End Base Document Definitions-->
</documents>

<!--Begin Derived Document Definitions-->
<documents name="xx">

  <!--Begin employeedetail document-->
  <document name="employeedetail"  table="Employee">
    <attribute name="Id"      field="ID"/>
    <!--Begin reference to departments schema-->
    <document name="Department"  docname="departments"
    field="Department_ID"      table="Department"
    joinfield="ID"             joinvalue="DepartmentID"/>
    <!--End reference to departments schema-->

  <!--End employeedetail document-->
  </document>

<!--End Derived Document Definitions-->
</documents>

```

XML Output

The Archway Document Manager will produce XML output with the following structure. The content of the XML elements will vary depending on the actual user record selected and the structure of the schemas referenced.

```
<employeedetail>  
  <Id>17156</Id>  
  <departments>  
    <Department>  
      <DepartmentName>Administration</DepartmentName>  
    </Department>  
  </departments>  
</employeedetail>
```

Frequently Asked Questions

This section contains answers to frequently asked questions about schemas.

How do I create document queries to linked tables?

To access data from linked tables, you must create a schema with nested `<document>` element entries.

You will need to create one `<document>` element for each table. Generally this will require creating one `<document>` element for the parent table and one `<document>` element for the target lookup table.

The following schema entries provide a template to use nested `<document>` entries to query linked tables. This template uses the in-place document nesting method.

```
<!--Base Document Definitions-->
<documents name="base">
  <document name="schema name">
    <attribute name="Id"      type="id"/>

    <document name="nested document">
      <attribute name="target XML tag" type="string"/>
    </document>
  </document>

<!--Derived Document Definitions-->
<documents name="adapter name">
  <document name="schema name"    table="parent table">
    <attribute name="Id"          field="primary key"/>

    <document name="nested document"  field="lookup field"
      table="lookup table" joinfield="Id field from parent table"
      joinvalue="Id field from lookup table">
      <attribute name="target XML tag" field="target field"/>
    </document>
  </document>
</documents>
```

Refer to the following table for an explanation of the schema entries used in the Derived Document Definitions.

Element and attribute	Description
<document name="schema name" ...	Since this is the first <document> element, the name attribute must match the schema name.
<document ... table="parent table">	The first <document> element describes the parent table. The parent table is the table that contains a lookup field whose value is determined by an entry in a lookup or link table.
<attribute name="Id" ...	The parent table must contain an <attribute> element named Id that provides the primary key field of the table.
<attribute ... field="primary key"/>	The parent table must contain an <attribute> element named Id that provides the primary key field of the table.
<document name="nested document" ...	Since this is the second <document> element, the name attribute can be any valid text string. Archway will use the name entered here as the name of the nested document XML tag.
<document ... field="lookup field" ...	This attribute defines what field in the parent table contains the lookup or link to the target table.
<document ... table="lookup table" ...	This attribute defines the target lookup table. The lookup table is the table that contains the actual database record you want to query.
<document ... joinfield="Id field from parent table" ...	This attribute defines the identifying field used in the parent table. This attribute tells Archway what record in the parent table has the lookup field to be queried.
<document ... joinvalue="Id field from lookup table">	This attribute defines the identifying field used in the lookup or link table. This attribute tells Archway what record in the lookup table contains the information to be queried.
<attribute name="target XML tag" ...	This attribute defines what XML tag will be generated by Archway as part of the document query.
<attribute ... field="target field"/>	This attribute defines the target field of the database record to be queried.

Why do all query response messages contain ID elements?

If you set the display form information option, you can see the XML output generated by your ECMAScripts. All document messages from the Document Manager will contain an <Id> element even if no such element was defined in the query used by the script. This is expected behavior from the Document Manager.

Archway creates a unique ID for each XML record to support back-end databases that use and create unique records. The ID element is used to prevent Archway from overwriting an existing database record and to speed up record searches.

All schemas need to have an <attribute> named Id that maps to the primary key of the database table. This schema entry allows Archway to uniquely identify each record. If Archway does not find an Id mapping in the schema, it will create one and search the back-end database for an Id field.

Can schemas include SQL statements?

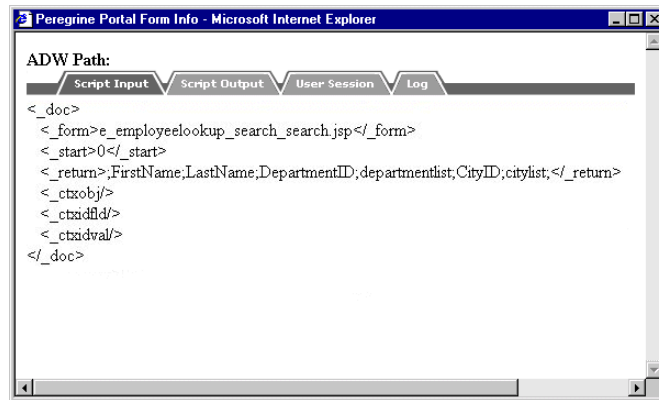
Schemas cannot contain raw SQL statements, but you can create schema entries that the Archway servlet will convert into SQL queries, updates, or insertions using the schema grammar described in "Using Nested <Document> Elements to Call Linked Tables" on page 134.

If you want to use raw SQL statements in your Web application, you can include them as part of ECMAScript messages. One method to include SQL queries in your schema document queries is to use either the *preprocess* or *postprocess* schema attributes to invoke an ECMAScript.

How are schema definitions converted into SQL statements?

The Document Manager converts ECMAScript messages into SQL statements at run time. Archway reads the XML document created when the form is loaded and uses the standard input tags plus any additional information passed when the form is submitted to generate a SQL statement. The Archway servlet will format the output of the SQL statement as an XML document. This XML document is then returned to the Web application ECMAScript for any additional processing and formatting.

For example, consider the search form of the example Employee Lookup Web application used in this guide. The input message generated is illustrated here.



Archway uses the values listed in the `<_return>` element to perform a Document Manager query. In this case it will search for the following:

Fields

- FirstName
- LastName
- DepartmentID
- CityID

Schemas

- departmentlist
- citylist

The Document Manager uses this XML document information to construct two SQL queries, one for each schema. The Log tab of the Form Info window would show the following two entries:

Executing Document Search:departmentlist

JDBCAdapter: Executing JDBC query: SELECT ID,DepartmentName FROM Department (0,-1)

Executing Document Search:citylist

JDBCAdapter: Executing JDBC query: SELECT CityID, City FROM City (0,-1)

The fields in the SELECT section of the SQL statement are determined by the <attribute> entries in the derived schema. The table listed in the FROM section of the SQL statement is determined by the <document table> entry in the schema.

Archway formats the results of these two queries into the single XML document listed below. This document will be listed in the `archway.log` if you have chosen to display form information in your Web application.

```
<_doc>
<recordset _count="-1" _countFound="19" _more="0" _start="0">
  <departmentlist>
    <Id>1</Id>
    <DepartmentName/>
  </departmentlist>
  <departmentlist>
    <Id>2</Id>
    <DepartmentName>Administration</DepartmentName>
  </departmentlist>
  <departmentlist>
    <Id>3</Id>
    <DepartmentName>Administrative Services</DepartmentName>
  </departmentlist>
  <departmentlist>
    <Id>4</Id>
    <DepartmentName>Burbank Agency</DepartmentName>
  </departmentlist>
  <departmentlist>
    <Id>5</Id>
    <DepartmentName>Burbank Unit</DepartmentName>
  </departmentlist>
  <departmentlist>
    <Id>6</Id>
    <DepartmentName>Deliveries</DepartmentName>
  </departmentlist>
</departmentlist>
```

```

        <Id>7</Id>
        <DepartmentName>Finance</DepartmentName>
    </departmentlist>
<departmentlist>
    <Id>8</Id>
    <DepartmentName>Helpdesk</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>9</Id>
    <DepartmentName>I.S. Department</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>10</Id>
    <DepartmentName>Los Angeles Agency</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>11</Id>
    <DepartmentName>Los Angeles Unit</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>12</Id>
    <DepartmentName>Marketing Management</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>13</Id>
    <DepartmentName>Operations</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>14</Id>
    <DepartmentName>Production</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>15</Id>
    <DepartmentName>Research</DepartmentName>
</departmentlist>
<departmentlist>
    <Id>16</Id>

```

```

    <DepartmentName>Sales</DepartmentName>
  </departmentlist>
<departmentlist>
  <Id>17</Id>
  <DepartmentName>Storage-Packaging</DepartmentName>
</departmentlist>
<departmentlist>
  <Id>18</Id>
  <DepartmentName>Taltek</DepartmentName>
</departmentlist>
<departmentlist>
  <Id>19</Id>
  <DepartmentName>Technical Support</DepartmentName>
</departmentlist>
</recordset>
<recordset _count="-1" _countFound="3" _more="0" _start="0">
  <citylist>
    <Id>1</Id>
    <Name>Burbank</Name>
  </citylist>
  <citylist>
    <Id>2</Id>
    <Name>London</Name>
  </citylist>
  <citylist>
    <Id>3</Id>
    <Name>Santa Clara</Name>
  </citylist>
</recordset>
<_form>e_employeelookup_search_search.jsp</_form>
</_doc>

```


A

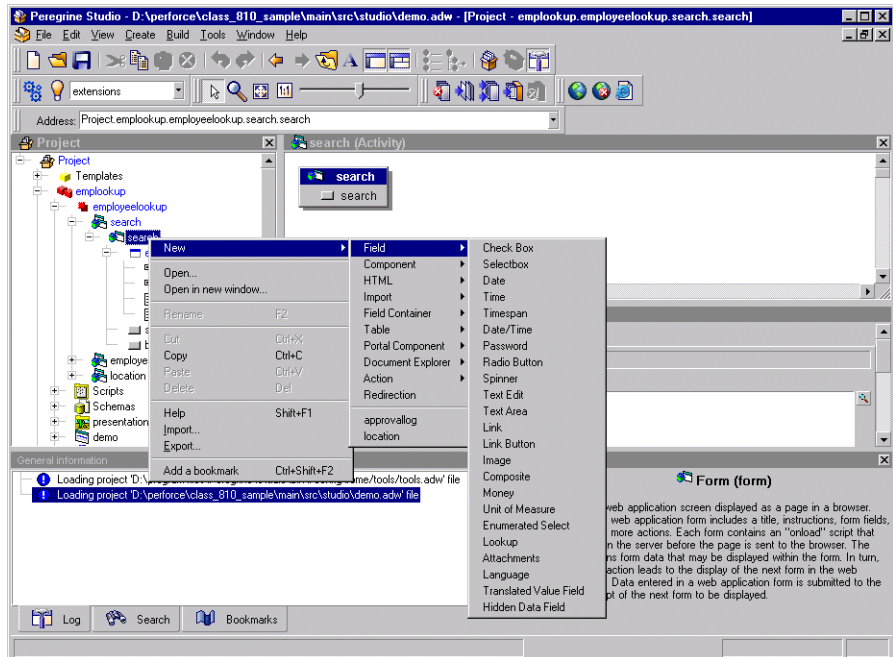
APPENDIX

Peregrine Studio Components

This appendix contains a list and description of all of the components you can add to a Project in Studio. The information is grouped according to the menu structure with which these components are presented in Studio, following each component down to display all of the subcomponents available.

The menus displayed when the package for your Web application is opened in Studio may vary slightly from the menu options documented here. Menu options change depending on the components you have created. For example, you must have the folder called `shared templates` in your package to enable DocExplorer Reference as a menu option.

To add components to your Project, right-click on the node to which you want to add a component, and a menu of options is displayed.



Project > New >
Directory Object—not supported.

Group of Modules > New >

When you create a Group of Modules component, it includes a folder called Explorers that contains default content for DocExplorer personalization screens. It also includes a Group of Roles, which is a list of roles that are used to control access rights. From the Group of Modules, you can create the following:

- **Module**—Web applications are organized into modules. Modules are often determined by the role that a user will take in performing tasks. For example, one module could be designed for employees who will be opening requests for service. Another module could be for managers approving requests. Modules are typically assigned specific access role restrictions so that only those users who need to perform the module's task have permission to do so.

- **The Peregrine Portal > Activity**—Each module should contain one or more activities that define the steps users can take to complete the module’s task. For example, a Request module could have activities for browsing catalogs, reviewing a shopping cart, and filling out a request form. Each activity is typically displayed in the Web application on a sidebar menu at the left of a form. Activities are typically assigned specific access role restrictions so that only those users who need to perform the activity’s task have permission to do so.

Form—Defines a Web application screen displayed as a page in a browser. The typical form includes a title, instructions, form fields, and one or more actions. Each form contains an onload script that executes on the server side before the page is sent to the browser. The script obtains form data that may be displayed within the form. In turn, each form action leads to the display of the next form in the Web application. Data entered in a form is submitted to the onload script of the next form to be displayed.

Field >

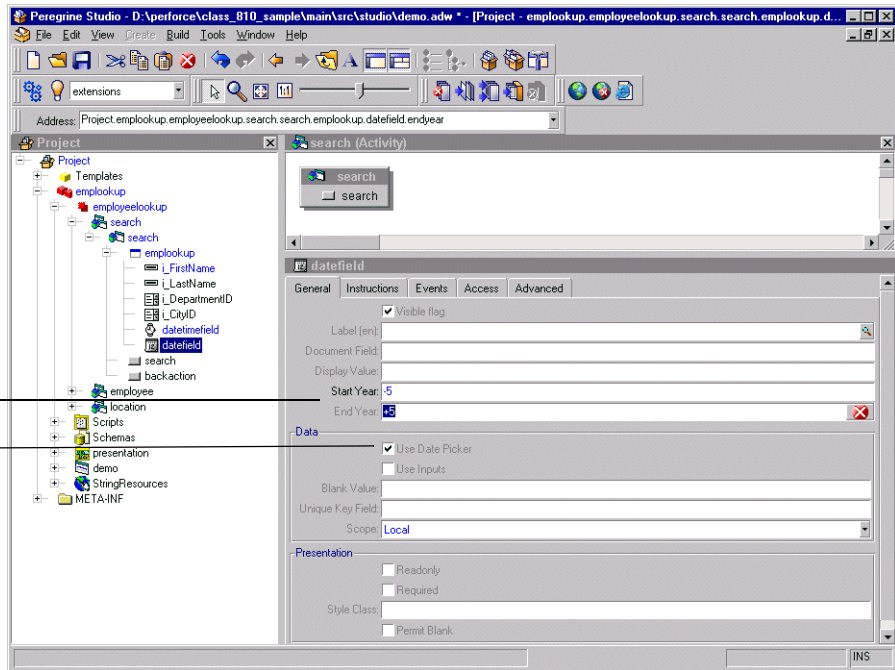
Check Box—Allows the user to toggle a value on or off.

Selectbox—Allows the user to select a value from a list displayed in a Combo Box field.

Date—Allows the user to view or enter a date. An optional calendar widget (Date Picker) can be enabled or disabled (the default is *enabled*). To define a start year for the drop-down list or for the calendar widget, add a + or - sign in front of a number. This number specifies the number of years before or after the current year you want the start and end years to be.

To designate a Date/Time calendar year start and end, add a + or - in front of a number to specify the number of years before or after the current year you want the start and end years to be.

Enable or disable the calendar widget.



Time—Allows the user to view or set a time value.

Timespan—Allows the user to view or edit a timespan value.

Date/Time—Allows the user to view or set a date and time value. There is an optional calendar widget (Date Picker) that can be enabled or disabled in Studio (the default is *enabled*). See Date component.

Password—Allows the user to enter a password.

Radio Button—Allows the user to select one of several choices presented by radio buttons.

Spinner—Allows the user to enter a numerical value. The control allows the number to be typed in directly. It also allows the user to select a number by clicking on the spinner buttons that increase and decrease the value.

Text Edit—Allows the user to display or edit a value in a plain text field.

Text Area—Allows the user to enter text into a multiline edit field.

Link—Displays a hyperlink that the user can click on to navigate to another Web location or site.

Link Button—Displays an image button created out of background images and text.

Image—Displays an image.

Composite—Allows the creation of a field that consists of two or more fields placed next to each other.

Money—Allows the user to view or edit a monetary value.

Unit of Measure—Allows the user to view or edit a value that is a unit of measure.

Enumerated Select—Allows the user to select a value from a list displayed in a Combo Box field.

Lookup—Allows the user to enter a value by performing a lookup operation. The lookup is done in a separate pop-up window.

Attachments—Allows the user to view and add attachments to a document.

Language—Allows the user to select their preferred language from a list of supported languages.

Translated Value Field—Displays text returned by a translation script function.

Hidden Data Field—Stores data obtained by the form's onload script without displaying it to the user. The data is included when the form is submitted and the user navigates to another form.

Component >

Treelink—Displays a treelink component.

Directory—Displays a directory component based on data received from a document query to an adapter.

List Builder—Allows users to configure a list by selectively adding items to a listbox from a list of choices.

Workflow—Displays a workflow diagram.

OAA Workflow—Displays a workflow diagram.

Stack—Displays a stack component.

SVG—Displays an SVG component.

Web Application Menu—Displays a menu of all registered modules or packages in the current Web application.

HTML >

Blank Line—Adds a blank vertical line to the form.

Free-form HTML—Allows you to insert arbitrary HTML tags into a form. Can also be used to insert client-side JavaScript into a Web page, although large amounts of JavaScript should be moved to a presentation file that can be imported by the page.

Import >

Static Import—Imports the text content of a file for inclusion in a Web page. For example, you can import files that define static HTML, JSP code or browser-side JavaScript functions.

External HTML Plugin—Includes dynamic content into the form. At run time, the URL referenced by the plugin is accessed by the server, returning contents which are then inserted into the form.

Field Container >

Field Section—Aligns fields into a column. Displays all field labels in an aligned column to the left of the fields. Fields can be divided into groups by inserting Headers and Instructions as needed. To display more than one column of fields, create a Form Columns container and place a Field Section container in each column.

Multicolumn Field Table—Organizes input fields into a multi-column table. It is recommended that you use FieldColumns and FieldSections instead.

Entry Table with Field Instructions—Organizes input fields into a multicolumn table with fields on the left and instructions for each field on the right.

Component Template—Allows you to define a group of form elements that can be reused in more than one form. Changes to the template are propagated to all places where the template is used.

Tabs—Adds tabs to a form, each pointing to different content defined by a separate form.

Dynamic Menu—Displays a multicolumn menu based on data received from a document query to an adapter.

Form Columns—Divides the form into columns, allowing content to be grouped and organized.

Table >

Simple Table—Displays a list of documents resulting from a query.

Document Table—Displays a list of documents resulting from a query.

Tree—Displays a list of documents resulting from a query as a tree.

Portal Component >

Component Editor—Generates fields elements used to configure a specific portal component. Not intended for general Web application use.

Portal Header—Generates the portal page header. Not intended for general Web application use.

Corkboard Header—Generates header information needed by any page that includes a corkboard. Not intended for general Web application use.

Corkboard Configurator—Generates a list of choices containing all known portal components. The list can be used to configure the components to display in a specific corkboard container.

Corkboard—Displays the portal components chosen and configured by each user.

Custom Configurator—Allows users to define their own custom component configurators.

Document Explorer >

Search—Displays a personalized list of fields used to perform document searches.

List—Displays a personalized table with the list of documents found as a result of a search.

Detail—Displays a personalized view of a document detail.

Action >

Action—Displays a button for an action. The button can be a link to another page or a submit action.

Default Action—Defines a form's submit action when no actual buttons are displayed.

Back—Navigates to the previous page of the Web application.

Home—Navigates to the home page of the Web application.

Print—Prints the current Web application form.

Close—Use to close pop-up windows.

Redirection—Redirects a page to a link depending on the result of the onload script matched against the condition

Transition—Contains an onload script and redirect arguments. After the script runs, execution is redirected according to the condition returned by the script. The options available from the Transition menu are the same as the Form menu, except there is no Action option.

- **Group of Strings**—List of multilingual strings.

Multilingual String—The name of the StringResource is the ID of the string.

- **Group of Scripts**—Server-side ECMAScripts.

- **Script**—Server-side ECMAScript (JavaScript) file containing functions used by Web application forms.

Header—Initial comments and imports required in this script file.

Function—Script function defining application logic executed on the server. All functions that have public access should accept a Message object as the single input parameter and return a Message object as a response. For example:

```
function xyz( msg ) {var msgResponse=new Message();...return
msgResponse;}
```

A script requires this public access interface if it is used as an onload script for a form or if it is called directly via an Archway HTTP message.

- **Group of Scripts**—Server-side ECMAScripts.
- **Group of Triggers**—A collection of triggers. Used by applications using BizDoc.
 - **Trigger**—Individual trigger for a document.
 - Message action**—Message action executed by the trigger.
 - Workflow action**—Workflow action executed by the trigger.
 - Script action**—Script action executed by the trigger.
 - Bizdoc Java action**—Java action executed by the trigger inside Bizdoc.
 - **Group of Triggers**—Collection of triggers.
 - Trigger**—Individual trigger for a document.
 - Group of Triggers**—Collection of triggers.
- **Group of Schemas**—Database schemas describing documents accessible by a Web application. Schemas define the field table mapping between the application and the back-end database.
 - **Raw Schema**—Description of a document's mapping on a real database.
 - **Schema**—not supported.
- **Group of Images**—Folder containing the image files to be used in your Web application.
 - **Image**—The image is loaded into the ImageData property as binary data. The file name property is used only the first time to load the image.
 - **Group of Images**—Folder containing image files.
 - Image**—The image is loaded into the ImageData property as binary data. The file name property is used only the first time to load the image.
 - Group of Images**—Folder containing image files.
- **Group of Presentation Files**—Folder containing files copied directly to the presentation folder for use within the Web application Web server.

- **Text Presentation File**—Any generic file in the Presentation folder that is needed by the Web server, for example, client-side JavaScript, static JSP files.
- **Binary Presentation File**—Binary file outputted in the presentation folder. Accessed by the Web server and used by the browser.
- **Group of Presentation Files**—Folder containing files copied directly to the presentation folder for use within the Web application Web server.
 - Text Presentation File**—Any generic file in the Presentation folder that is needed by the Web server, for example, client-side JavaScript, static JSP files.
 - Binary Presentation File**—Binary file outputted in the presentation folder. Accessed by the Web server and used by the browser.
 - Group of Presentation Files**—Folder containing files copied directly to the presentation folder for use within the Web application Web server.
- **Group of default DocExplorer screens**—Folder containing default content for DocExplorer Personalization screens.
 - **Reference of a file**—File object.
 - **Directory Object**—not supported.
- **Group of Portal Components**—Components that appear in the portal components menu and can be added to the home page by the user.
 - **Portal Component**
 - (**contents**)—The content of the portal component that is displayed.
 - (**configure**)—Allows configuration of a portal component.
- **Group of Files**—A temporary container of miscellaneous files used by a Web application. For example, string files and `scriptpoller.ini` files are stored here.
 - **String file**—Temporary representation of a string file.
 - **Ini file**—Temporary representation of a `scriptpoller.ini` file.
- **Group of Strings**—List of multilingual strings.
 - **Multilingual String**—The name of the StringResource is the ID of the string.
- **Group of Roles**—not supported.

Group of Style Sheets > New >

- Style Sheet—Not supported.

Group of Roles—not supported.**Group of Files > New >**

- String file—Temporary representation of a string file.
- Ini file—Temporary representation of a scriptpoller.ini file.

Group of Strings > New >

- Multilingual string—The name of the StringResource is the ID of the string.

Entities (collection of business objects) > New >

- Entity—Used by applications using BizDoc.

Interfaces > New >

- Interface—Not supported.

System enumerations > New >

- System enumeration—Describes a system enumeration, used to define data attributes where the value stored is not the value displayed to the user. This allows multilingual databases.
 - Value—Defines one value for a system enumeration.

Templates > New

- Schema >Not supported.
- Field Container
 - Component Template
- Directory Object—not supported.
- Group of Methods—Includes a list of methods. You can create new methods under this element.
 - Method—Java Method. The name is not significant. You can add a comment to the method.
- Method—Java Method. The name is not significant. You can add a comment to the method.
- Message action—Message Action executed by the trigger.

- **Workflow action**—Workflow action executed by the trigger.
- **Bizdoc Java action**—Java action executed by the trigger inside Bizdoc.
- **Script action**—Script action executed by the trigger.
- **Trigger**—Used by applications using BizDoc.
- **Group of Images**—Allows you to create a group of images.
- **Attribute**—Ejb attribute. Used by BizDoc.
- **Reference**—Ejb reference. Used by BizDoc.
- **Contain**—Contain an object as an embedded member.
- **Computed**—Computed property.
- **Structure**—Ejb structure. Used by BizDoc.
- **Collection**—Ejb collection. Used by BizDoc.
- **Methods**—Ejb method. Used by BizDoc.
- **Entity**—Ejb entity. Used by BizDoc.

B

Troubleshooting and FAQs

CHAPTER

This chapter contains troubleshooting information for Peregrine Studio and Web application tailoring.

Web Application Environment

This section describes warnings or errors that can be generated while running a Peregrine Web application in your system environment.

Out of memory error

Problem

Your application server has run out of memory resources.

Solution

Peregrine Web applications run best on a system with a minimum of 512 MB of RAM. If you cannot add more physical memory to your machine, you can increase the virtual memory space used on your Windows system. Adding virtual memory will require more hard disk space and may degrade system performance as cached information is saved to and retrieved from the hard disk. Refer to your Windows help for information on setting or changing virtual memory.

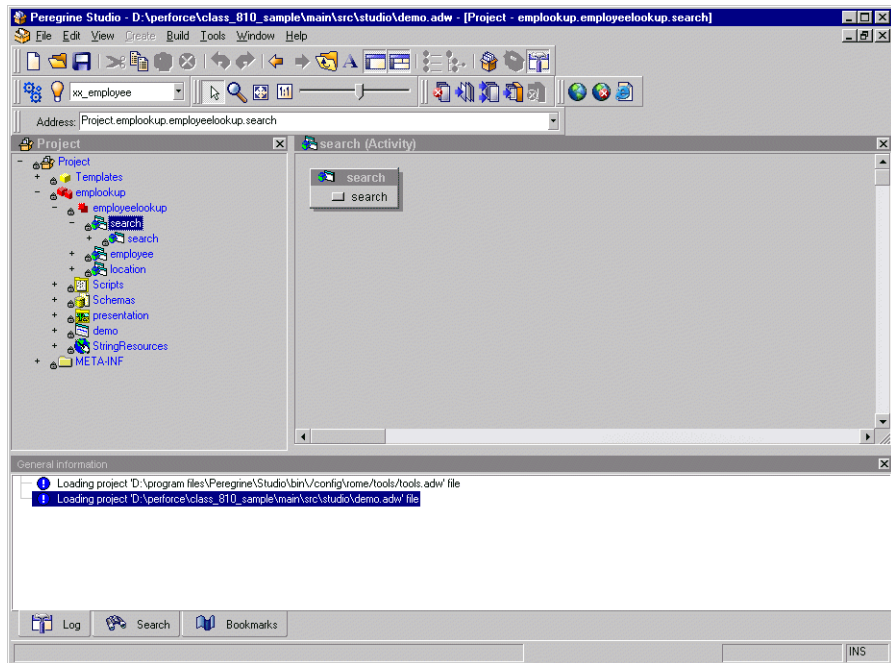
Peregrine Studio

This section describes common problems with write protections, conflicts, and build errors generated with Studio.

Cannot edit — components are displayed with grey background

Problem

Studio displays some or all of your project components with a grey background, and you cannot make or save changes to the project components.



Solution

Studio uses the grey background to indicate that an item is write protected. The most common reasons that Web application components are write protected are:

- A write-protected package is selected in the package selector.
- The project (.adw) file is set to read-only.

Packages delivered by Peregrine are write-protected. You must save all of your changes and additions to a user-created package, extensions. If the package selection box displays one of the Studio default packages, then your project will be write protected until you create and activate a new package extension in which to save your changes.

Red exclamation point (conflict icon) displayed next to nodes


Problem

Studio displays a conflict icon next to one or more of your Web application components, and you cannot build the project. The conflict could be the result of multiple packages attempting to change or modify the same component, or the conflict could be the result of improperly defined package dependencies.

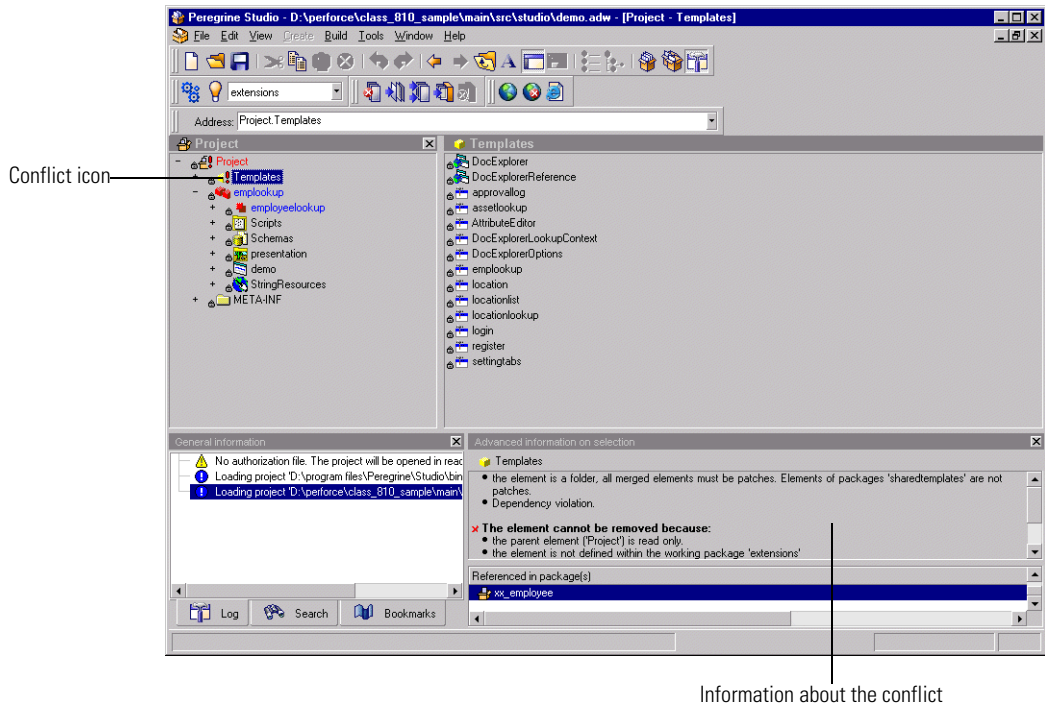
Solution

To resolve the conflict you should first view more information about the nodes displaying the conflict icon.

To view information about a conflict:

- 1 Select a node with an exclamation point  icon displayed next to the name from the Project Explorer view.
- 2 Click View > Advanced Information. Studio displays a new information window at the bottom of the interface. This window displays information on the conflict.

The information on selection will tell you whether you have a resource or a dependency conflict.



Resource conflicts

Resource conflicts occur when two or more project components describe the same thing. To resolve a resource conflict, delete or reconfigure one of the project components that is creating the conflict. If the conflicting components are part of separate package extensions, you can choose to deactivate one of the package extensions to resolve the conflict.

Dependency conflicts

Dependency conflicts occur when a package extension attempts to modify a package that is not listed as a dependent package. To resolve the conflict you can choose one of two solutions:

- Add the package you want to modify as a package dependency of the conflicting package extension.
- Move the changes in the conflicting package extension to another package extension that already has the proper package dependencies.

Import Errors

Web applications built on Peregrine OAA Platform 2.2 use a different XML data structure than previous releases of Get.It!. As a result of the new XML data structure, you can use the import nodes feature only for nodes that you created in and exported using Studio. Get.It! 1.3 and 2.1 customizations can be imported as new Studio projects using the import wizard.

Unable to import customized files

Problem

Cannot find a means to import customized Get.It! 1.3 XSL files.

Solution

Studio does not support customized XSL files. Contact your Peregrine sales representative for additional information.

Bad magic number

Problem

This error occurs only if you are using JRun as your application server. A *500 Internal Server Error* (Bad Magic Number) error page is displayed instead of the login screen.



Solution

Clear both the JRun Default Server and the JSP cache, and then stop and restart the JRun Default Server.

To clear the JRun JSP cache:

- 1 Open Windows Explorer.
- 2 Browse to the following location:

<drive letter>:\Run\servers\default\oaa\Web-inf\jsp

- 3 Delete all files in this folder.

Scripting Errors

Information about scripting errors is displayed as text at the top of the main frame and in the `archway.log` file.

Cannot find script file

Problem

The following error message is displayed when you select a form:

Unable to find script file for <name>

This message will also appear in the `archway.log` file.

Solution

This error message is usually the result of a script file trying to call an undefined adapter. This is a common problem if you import a Web application into a project that contains a new adapter. Review your script file and determine what adapters it calls. If the <name> value is the name of a new adapter defined in the script file, then define the new adapter in the Admin Settings module, stop and restart your application server, and then restart the Archway server (using the Admin Control Panel) to correct the problem.

This error message can also appear if a form is calling an invalid script file name. Verify in Studio that the form is calling a valid script file name. If you copied a script from another form or Web application you may have renamed the script incorrectly.

If you have verified that the script file exists and uses the proper adapter, then stop and restart your application server. This will refresh the adapter settings.

Script produces an ECMAScript error

Problem

An ECMAScript Error is displayed with the script name, source code, and line number of the error when a form is displayed.

Solution

Open Studio, review the error-producing script for typos, and verify that it uses the correct function and schema names. For example, in the form above, the function *msg* is incorrectly listed as *nsg*. Correct any errors and rebuild the project.

Note: ECMAScript is case sensitive and will return an error message if the case does not match the object called.

Tip: If you have enabled the HTTP listener in Studio, you can click on the underlined script name listed at the top of the error message to go directly to the script and line number of the error. Studio must be open for the hyperlink to work.

ECMAScript error: undefined value or property

Problem

The following error is displayed when you select a form:

```
ECMAScript Error: Error Message: Runtime error Function called on undefined value or property
```

This error will also be displayed in the `archway.log` file.

Solution

Verify that the form calls the proper script name in the server onload script attribute. Also check that the script name contains no typos and that it is listed with the proper case. If the script name listed in the form is correct, there is a possibility that there is a script name conflict. Each script in your project needs a unique name. Try renaming your script to a new name, updating the server onload script attribute, and rebuilding your project. If renaming the script fixes the problem then you had a script name conflict.

Web Application Errors

The following sections describe some of the common errors associated with custom-made Web applications. Refer to the sections below for solutions to common Web application design problems.

Wrong start form is displayed for activity

Problem

You want your Web application to display a particular form when you select an activity, but the wrong form is displayed. You may have also re-ordered the form listing in the Project Explorer tree, but the proper form still is not displayed.

Solution

You need to define the Start Page attribute of the activity. This attribute determines what form is first displayed when the activity is selected. By default, the Start Page is blank.

To set the Start Page of an activity:

- 1 Open Studio and select the activity you want to change.
Tip: To select the activity properties, select the activity node, double-click any form in the flowchart view displayed, and then click the Control tab. The activities properties will be displayed to the right of the control flowchart.
- 2 In the properties of the activity, use the selectbox of the Start Page attribute to choose a starting form.
- 3 Save and rebuild your project file.

Script output not appearing in form component

Problem

Data is not displayed in your Web application form component. This problem could be the result of a faulty script that is not generating an XML document or the result of form components that are not properly mapped to the fields of the generated XML document.

Solution

Verify whether your script is generating an XML document by enabling the **Show form information** option and then looking at the contents of the Script Output tab. If the script is working properly, you should see your Web application data encoded as in the XML document displayed on the Script Output page. If you do not see an XML document, then your script has an error.

If you can see data displayed in the Script Output tab, then the problem is how you have mapped the form components to the XML fields. View the form component properties from Studio, and verify that the Document Field attribute of the form component maps to an XML tag displayed in the Script Output tab.

Too few parameters error

Problem

The following error message is displayed when you select a form:

```
ERROR:jdbcCalls: ***SQL Exception caught***
```

The script output displays the following error:

```
-3010: [Microsoft][ODBC Microsoft Access Driver] Too few parameters. Expected 1.
```

These messages will also appear in the archway.log file.

Solution

There is an incorrect field mapping or typo in the schema used in this form. Review the schema(s) used by this form and verify that there are no typos. Also verify that all the attributes defined in the schema map to valid fields in the back-end database. The value in the field attribute must match the field name of the back-end database. This is particularly important for the ID attribute, which must map to a unique numerical value that identifies each record.

Web application always goes to redirection form

Problem

You have defined a redirection to another form in your Web application and the application always takes users to the redirection form regardless of the search conditions and results.

Solution

Validate that the Condition attribute of the redirection is not blank. The Condition value should match the value defined by the setCondition function of your form's ECMAScript. If the Condition attribute is left blank, the default action is to redirect to the target form regardless of the returned results.

JDBCCalls error at login

Problem

The following error is displayed when you login:

```
ERROR: jdbcCalls ***SQL Exception caught***
```

This error will also be displayed in the archway.log file.

Solution

The JDBC adapter you have created is trying to authenticate users. Turn off authentication for this adapter and then the error message will disappear.

Syntax error in FROM clause

Problem

The following error message is displayed when you select a form:

```
ERROR:jdbcCalls: ***SQL Exception caught***
```

The script output displays the following error:

```
-3506 [Microsoft][ODBC Microsoft Access Driver] Syntax error in FROM clause.
```

This error will also be displayed in the archway.log file.

Solution

The schema name you defined for the form is wrong. The schema name could be listed incorrectly in two places:

- The form's onload script may refer to the wrong schema name.
- The <document name=value> does not match the schema file name.

Index

A

- actions. See form components
- activity component 34
- Archway
 - schemas and 118
 - scripts 93
 - XML output 143
- AssetCenter 80
- AssetCenter adapter
 - setting feature links 122
- authorization file 19

B

- Bad Magic Number error 165
- BizDoc 80
- bookmarks, adding in Studio 26
- build options, setting 41
- build process, files created during 39

C

- Cascading Style Sheets 32
- character encoding
 - setting in Studio 40
- component template 69–70
- components
 - build process 39
 - defined 32
 - group of files component 35
 - group of modules component 34
 - group of schemas component 35
 - group of scripts component 35

- hierarchy of 33
 - module component 34
 - relationships among 34–35
- components in Studio 149
- conflicts
 - defined 45
 - resolving 45–46, 164
- creating
 - nested document lookups 87
 - package extensions 44

D

- Date component, defining start year 152
- Date Picker 152
- dependencies, setting package 43
- dependency conflicts. See Conflicts
- Deployment directory 40
- development environment, requirements for 20
- DocExplorers
 - adding as a Reference 83
 - personalizing with 82
- document schema definitions. See schemas

E

- ECMA scripts
 - defined 32
- ECMAScript 92
- ECMAScripts
 - examples of 105–114
- errors
 - Bad Magic Number 165

Cannot Find Script File 166
import 165

F

feature links in AssetCenter, setting 122
field labels, changing 60
Field Lookup. See Lookup fields
fields. See form components
fieldsection component 70
form component 34
form components
 action 35, 77–78
 changing schemas 62
 component template 69
 described 35
 document field names 64
 field form components 59
 fields 35
 fieldsection 70–71
 hidden data field 74
 hiding 60
 labels 59
 lookups 35
 making read-only 61
 names in 64–66
 redirection 74–75
 selectbox 72–73
 simple table 75–76
 table link 76
 tables 35
 tailoring 54–78
 text columns 77
 text edit 71–72

forms

changing instructions 56
changing onload scripts 57
changing titles 55
Personalization and 89
server-side 94–95

FTP

configuring Studio 47
deploying Web applications to UNIX 46

G

group of scripts component 35

H

HTTP Listener 28

I

installing 14
instructions, changing in forms 56
interface components. See Form components 35
international builds 48
ISO character encoding. See character encoding

J

Java Virtual Machines 104
JavaDocs 115
JavaScript 92
JVMs 104

L

lookup fields 85
 creating 85
 nested document lookups 87
lookups. See form components

M

messages, scripts 100

N

nodes 27, 163

O

onload scripts
 changing in forms 57
 defined 57
opening source files in Studio 19

P

package extensions 44–??
packages
 activating 42
 deactivating 42
 defined 41
 dependencies 43
Personalization
 hierarchies 82
 interface, described 88
 lookup fields 85

- modifying forms 89
- requirements 80
- settings 80
- user rights 80
- presentation files 32
- Project Explorer 27
- projects
 - See also Web applications
 - components of 32
 - conflicts within 46
 - files within 38

R

- reference, adding a DocExplorer 83
- resource conflicts. See conflicts

S

- schemas
 - Archway and 118
 - attributes 123–140
 - changing in form components 62
 - defined 117
 - document fields 63
 - document queries 141
 - elements 123–140
 - frequently asked questions 141–??
 - schema entries 133
 - SQL statements 143
 - testing from URL 96–97
 - using with DocExplorers 83
- scriptpollers.ini, adding an entry 103
- scripts
 - adding to script pollers 112
 - client-side 92
 - ECMAScript 92
 - enabling script pollers 102
 - JavaScript 92
 - JVMs and script pollers 104
 - onload scripts 94–95
 - roles of 93
 - script pollers 102–104
 - server scripts 91, 93
 - server-side 92
 - stopping script polling 104
 - testing onload scripts 95–96

- XML messages 100
- ServiceCenter 80
- source code, viewing 30
- source files, opening in Studio 19
- string files
 - exporting 49
 - importing 51
 - translating 50
- Studio
 - authorization file 19
- Studio components 149

T

- tables. See form components
- tailoring
 - form components 54–78
- templates component 34
- Temporary directory 40
- testing environment, requirements for 20
- titles, changing in forms 55
- translation strings. See string files
- troubleshooting
 - conflicts 163
 - import errors 165
 - Read-only components Studio 162
 - script errors 167
 - virtual memory error 161
 - Web application errors 168–170

U

- UNIX
 - deploying Web applications to 46
- URL
 - querying from 95
- user rights, Personalization 80

V

- variables
 - format 97
 - referring to XML attributes 100
 - using in form components 97
- visible flag
 - hiding form components 60

W

Web applications

- components. See Components
- described 10
- hierarchy and 36
- scripts and 91
- tailoring forms 54–68
- viewing source code 30
- viewing changes 30

X

XML

- Archway 143
- components. See components
- form component field names 65
- messages in scripts 100
- using in variable names 97
- viewing source code 30

