User's Guide

iPlanet Application Builder

Release 6.0

806-4997-01
May 2000

Contents

Preface

The *iPlanet Application Builder 6.0 User's Guide* describes how to use the application builder tool. This manual is intended for the software developer, but can be used by anyone involved in application development; including application integrators, administrators, and release engineers.

This manual assumes some understanding of the iPlanet Application Server™, the Solaris™ operating environment and UNIX® commands, and the Windows NT operating system.

Multiplatform Release

This iPlanet Application Builder 6.0 release supports versions 2.6 and 8 of the Solaris ™ SPARC™ *Platform Edition*, the Solaris *Intel Platform Edition*, and the Windows NT operating environments.

How This Book Is Organized

Chapter 1, "Desigining Applications," presents the design logic and application planning needed in developing a well designed application.

Chapter 2, "Setting Application Builder Options," describes how to set and change the iPlanet Application Builder options.

Chapter 3, "Creating and Managing Projects," provides step-by-step instructions on how to create, edit, and build a project.

Chapter 4, "Creating Data Access Logic," describes the concepts and tasks for creating a model of your data source.

Chapter 5, "Creating Presentation Logic," describes the concepts and tasks associated with creating .the presentation logic for web aplications and describes how the use the Java Editor and Properties window to customize servlets.

Chapter 6, “Creating Presentation Layout,” describes how to create the pages that make up your application user interface.

Chapter 7, “Creating Business Logic.describes how to create Enterprise Java Beans.

Chapter 8, “Using Wizards to Generate Project Files,” describes the code generation wizards that help you perform common tasks associated with building an iPlanet application.

Chapter 9, “Compiling, Testing, and Debugging Applications,” gives a brief overview of the steps required to successfully build, test, and debug your application.

Chapter 10, “Deploying and Downloading Applications,” describes deploying the finished product onto one or more iPlanet Application Servers.

Chapter 11, “Working with Source Control,” describes how to configure the source code control and share the settings among your programmers.

Appendix A, “Component Reference,” identifies each of the reusable components available with the application builder.

Appendix B, “Using the Application Builder with Third-Party Tools,” describes the interoperability considerations for using the application builder with third-party tools.

Appendix C, “Using Existing Projects with iPlanet Application Builder,” describes how you can use projects from an earlier application server release with the application builder 6.0 release.

The Glossary defines terms that may be unfamiliar to application builder users.

The graphic images in this manual are on the NT platform. UNIX image displays may appear slightly different.

Typographic Conventions

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

URLs are used in the form http://*server.domain*/*path*/*file*.html

Where:

- *server* is the name of the server where you are running the application.

- *domain* is your internet domain name.
- *path* is the directory structure on the server.
- *file* is an individual filename.

The following table shows the typographic conventions used throughout the iPlanet documentation.

Table 1 Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123`	The names of files, directories, sample code, and code listings; and HTML tags	Open `Hello.html` file. `<HEAD1>` creates a top heading.
AaBbCc123	Book titles, variables and placeholders, words to be emphasized; words used in the literal sense	Read Chapter 8of the *Programmer's Guide*. Enter your *UserID*. Enter *Login* in the Name field.
AaBbCc123	glossary terms	**Templates** are page outlines.

Related Documentation

You can access documentation about the iPlanet Application Server (iAS) and iPlanet Application Builder (iAB) in the following ways:

- Through the Internet at the `iplanet.com` Web site at: `http://docs.iplanet.com/docs/manuals`
- Through the installed iPlanet Application Server on your local system or network. Open the `installdir/ias/docs/index.htm`, where `installdir` is the directory in which you installed the iPlanet Application Server. From there, you can locate the documentation for the iPlanet Application Builder.

The following table lists related iPlanet manuals by the task you are trying to perform.

Table 2 Related iPlanet Manuals

For information about	See the following	Shipped with
Late-breaking information about the software and documentation	`readme.html`	iAS 6.0 Developer Edition

Table 2 Related iPlanet Manuals *(Continued)*

For information about	See the following	Shipped with
Installing the Application Server and its components, and configuring the sample applications	Installation Guide	iAS 6.0
Installing the Application Builder	`install.html`	iAB 6.0
iAs features , such as the components, general capabilities, and system architecture	Overview	iAS 6.0
Deploying the Application Server at your site, by performing the following tasks: • Planning your Application Server environment • Integrating the product within your existing enterprise and network topology • Developing server capacity and performance goals • Running stress tests to measure server performance • Fine-tuning the server to improve performance	Deployment Guide	iAS 6.0
Administering one or more application servers using the Application Server Administrator tool to perform the following tasks: • Deploying applications with the Deployment Manager tool • Monitoring and logging server activity • Setting up users and groups • Administering database connectivity • Administering transactions • Load balancing servers • Managing distributed data synchronization	Administration Guide	iAS 6.0
Migrating your applications to the new iPlanet Application Server 6.0 programming model from version 2.1, and 4.0; including a sample migration of an Online Bank application provided with the Application Server	Migration Guide	iAS and iAB 6.0

Table 2 Related iPlanet Manuals *(Continued)*

For information about	See the following	Shipped with
Creating iAS 6.0 applications within an integrated development environment by performing the following tasks: • Creating and managing projects • Using wizards • Creating data-access logic • Creating presentation logic and layout • Creating business logic • Compiling, testing, and debugging applications • Deploying and downloading applications • Working with source control • Using third-party tools	User's Guide	iAB 6.0
Creating iAS 6.0 applications that follow the J2EE standard (Servlets, EJBs, JSPs, and JDBC), by performing the following tasks: • Creating the presentation and execution layers of an application • Placing discrete pieces of business logic and entities into Enterprise Java Bean (EJB) components • Using JDBC to communicate with databases • Using iterative testing, debugging, and application fine-tuning procedures to generate applications that execute correctly and quickly	Programmer's Guide (Java)	iAS 6.0
Using the public classes and interfaces, and their methods in the iPlanet Application Server class library to write Java applications	Server Foundation Class Reference (Java)	iAS and iAB 6.0
Creating iAS C++ applications using the iAS class library by performing the following tasks: • Designing applications • Writing AppLogics • Creating HTML templates • Creating queries • Running and debugging applications	Programmer's Guide (C++)	Order separately

Table 2 Related iPlanet Manuals *(Continued)*

For information about	See the following	Shipped with
Using the public classes and interfaces, and their methods in the iPlanet Application Server class library to write C++ applications	Server Foundation Class Reference (C++)	Order separately

Chapter 1

Designing Applications

This chapter describes the relationships between files used in creating applications to run on the iPlanet Application Server.

The following topics are described in this chapter:

- Parts of an Application
- Assembling the Pieces
- Getting Started

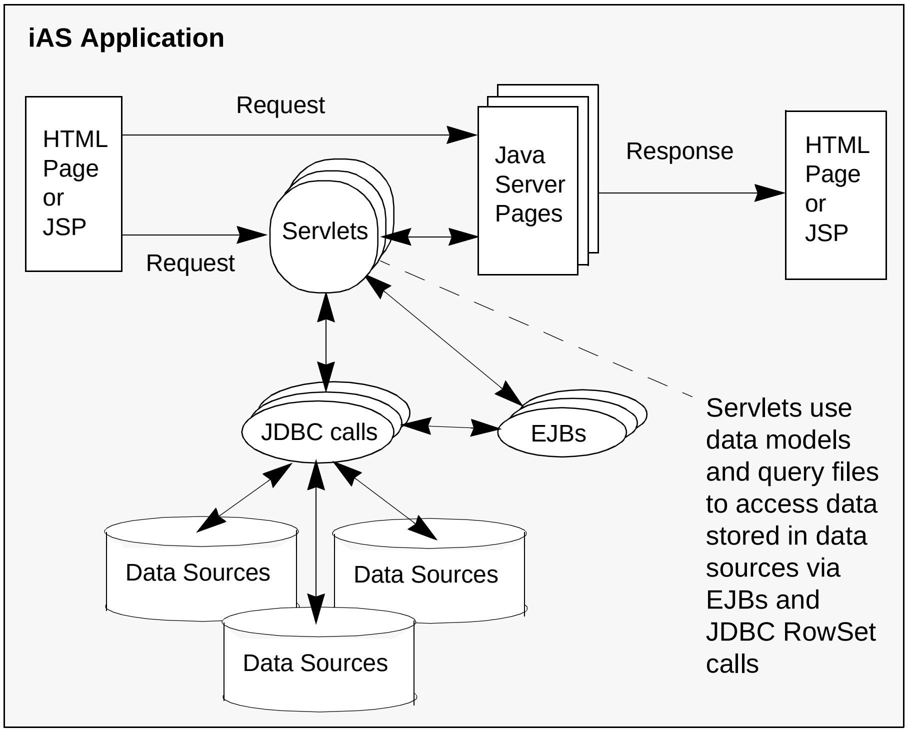
Parts of an Application

In the iPlanet Application Server environment, an application can contain three types of programming layers. As the developer, you use or create objects to deliver each specific layer, as listed in the following table:

Table 1-1 Application Layers

Application Layer	**Objects Used**
Data access layer	Query files, and data models
Presentation layer	Java servlets for the presentation logic; JavaServer Pages and HTML files for the presentation layout.
Business layer	Enterprise Java Beans

The relationships among the objects which represent the various application layers are shown in the following figure and are explained in the three sections that follow.



Data Access Layer

Data access logic determines what types of back-end data sources (if any) the application accesses, such as a relational database or a legacy system. In this guide, it is assumed that a data source already exists for use with your application, and that there is someone at your site, such as a database administrator, who manages it. iPlanet Application Builder enables you to access your data sources via data models and data access query files.

To legacy systems, your code may need to call into an extension. iPlanet Application Server includes prebuilt extensions. These extensions allow an application to communicate with transaction-processing systems such as CICS/IMS, IBM MQ Series, and BEA Tuxedo. Additional prebuilt extensions allow communication with ERP systems such as SAP R/3. If you need to create a custom extension, use iPlanet Extension Builder.

Relational Data Sources

If the application accesses a relational database, you use queries to access the data. To use a query, create two types of files:

- a **data model**, which specifies the database tables and relationships used by the application.
- a **query file**, which contains one or more queries, each of which defines a single interaction with a relational database.

You cannot create a query without a data model. Every query file depends on a data model, but not necessarily the same data model.

Nonrelational Data Sources

If the application is required to access a nonrelational data source, you must write Java code to communicate with the data source. In some cases, you can call into an extension that is prebuilt into iPlanet Application Server. In other cases, you may want to build a custom extension using iPlanet Extension Builder.

Presentation Layer

The presentation layer handles the presentation logic and layout of the application. The following sections introduce these concepts.

About Presentation Logic

Presentation logic controls and processes the data generated from application users by invoking Enterprise JavaBeans to perform business logic functions, and then generating a dynamic page that sets up the next user interaction. Presentation logic is handled by servlets on the iPlanet Application Server. Servlets handle such tasks as page-to-page navigation, session management, simple input validation, and the tying together of business logic.

About Presentation Layout

Presentation layout determines how users experience the application. Typically they use a web browser and navigate from page to page (HTML pages or JavaServer Pages). For example, suppose a user is viewing the left web page in the previous figure. As a result of a user action, such as pressing a Submit button, the web browser displays the page on the right. The intervening processing remains transparent to the user.

Business Logic Layer

Business logic maintains the application-specific processing and business rules of an application. Using iPlanet Application Builder, you implement your business logic via Enterprise JavaBeans (EJBs).

The application components designed for your business logic connect the components designed for the presentation layer (servlets, HTML pages, and JavaServer Pages) to the components designed to implement the data access layer (data models, queries, and JDBC RowSet calls).

Enterprise JavaBeans

In the iPlanet application model, you design session and entity EJBs to implement the various types of business logic.

Assembling the Pieces

To create an application, you assemble files that execute the presentation logic, business logic, and data access logic that the application requires. Assembling files means either creating new files or locating existing files to add to a project, which is a container for your files during development. A project also lists the files you will deploy as an application, after development is completed. The process of creating projects with iPlanet Application Builder is described in “Creating and Managing Projects.”

Types of Files in a Project

In developing an application, you assemble the following types of files:

Table 1-2 File Types

File Description	**File Suffix**
HTML pages	`.html, .htm`
JavaServer Pages	`.jsp`
JavaScript	`.js`
Image files	`.gif, .jpg, .jpeg`
Servlets, EJBs, and other Java code	`.java`, `.class`
Query files	`.gxq`

Table 1-2 File Types

File Description	File Suffix
Data models	`.kdm`
Connection files	`.props`

How Files Relate to Each Other

You use iPlanet Application Builder to create a project that ties its files together. The files in a project are related as follows:

- HTML pages and JavaServer Pages (JSPs) contain the web content. In addition to entering formatted text, you can insert links to image files. You also design HTML pages and JSPs that call JavaScript, Enterprise JavaBeans, and servlets.
- Servlets contain the application's business logic written in Java. For example, a servlet can call another servlet or return data via a RowSet object. The servlet handles data access, which may involve calling a query file to execute a query against a relational database. And when a servlet returns data via a RowSet object, it merges this output with a JSP.
- If the application must query a relational database, you create one or more data models, and one or more query files based on a data model.

Although you can use iPlanet Application Builder to manually define the relationships between files, one of the benefits of iPlanet Application Builder is its automated code generation wizards. Wizards guide you in creating sets of related files to perform commonly needed actions.

For example, many applications require login functionality, which derives from the interaction of several files. Using the Login wizard, you can quickly create the needed files, with the links between them automatically defined. For more information on wizards, see "Using Wizards to Generate Project Files."

Getting Started

The general process of creating a project is described in "Creating and Managing Projects." However, before you launch iPlanet Application Builder, the following conditions must be in place:

- Understand the iPlanet Application Server Environment

- Define the Application Environment
- Define the Application Requirements
- Define the User Interface
- Set Up the Data Sources

Understand the iPlanet Application Server Environment

Because you will be deploying applications to the iPlanet Application Server, it is important that you understand the role of these applications within a three-tiered environment. Before using iPlanet Application Builder, it is recommended that you become familiar with the concepts presented in the *Programmer's Guide*.

Define the Application Environment

What type of application do you want to design? Possible variations include: Internet, intranet, and extranet. Some examples are shown in the following table:

Table 1-3 Application Types

Application Type	**Examples**
Internet	E-Commerce, Financial, Security, Portal, Internet Service Provider (ISP).
Intranet	Human Resources self-service, Sales force automation, internal help desk
Extranet	Supply chain management, insurance quotes, car dealership management
Workgroup	Accounting reports, RFE management

Define the Application Requirements

Once you've defined the application type, the next step is to gather the requirements of that application. Requirements are often determined by answering questions about the application's purpose or functionality. For example:

- Will users be anonymous, or will they be tracked by a username and password?

- Will the application support different user interfaces, such as multiple languages or different versions of a browser?
- What kinds of data can users query or update, and how is it accessed?
- Additional issues include: performance, throughput, capacity, scaling, reliability, and security.

The previous list is only a sample of questions to consider. For more information on gathering application requirements, see the *Programmer's Guide*.

Define the User Interface

Defining the user interface determines the page flow, which affects how an end-user navigates through the application. You define page flow by deciding what files will call other files. The general principle of page flow is shown in the previous figure.

The following questions can guide your decisions about user interface design:

- What is the page layout and flow for the application?
- What commands and buttons are available on each page?
- Will the pages use frames or not?
- Does the application require user validation?

  If so, you may decide to create a login page through which users access the application. Note that iPlanet Application Builder provides a Login wizard to guide you in creating login functionality.
- How will data be entered and displayed? In particular, the following questions pertain to this issue:
- How many HTML pages or JavaServer Pages require input forms?
- What kinds of form elements are required for the HTML pages or JSPs?
- Will database results be displayed together on a single page or displayed across multiple linked pages?
- Are there any corporate standards that determine headers and footers, logos, menu bars, or banner ads?
- Are there any international issues? For example, are the icons or cartoon images meaningful to all users? Does translation pose a formatting problem?

- Are the commonly used features easy to find?

iPlanet Application Builder provides several wizards that make it easy to create some of the more common page designs.

For additional information on application design, see the *Programmer's Guide*.

Set Up the Data Sources

A data source can be any of the following examples:

- relational database
- directory server
- legacy system (for example, mainframe files)
- client/server application (for example, Enterprise Resource Planning, or a spreadsheet)

Most web applications access a data source in some way; for example, when they display query results or when they allow users to update account information.

In the case of a relational database, a database administrator or data architect sets up and manages access at your site. Similarly for non-relational data sources, it is assumed that these systems are available and that they provide access to clients.

If there is one or more data sources for you to work with, you will need to specify which data source(s) to access, and which connection protocols to use. These tasks are described in "Creating Data Access Logic."

Chapter 2

Setting Application Builder Options

This chapter describes how to set and change the iPlanet Application Builder options.

The following topics are described in this chapter:

- About iPlanet Application Builder Options
- Setting Project Options
- Setting Development Options
- Setting Runtime Options
- Customizing the Palette

About iPlanet Application Builder Options

In iPlanet Application Builder, you can set options to customize your work environment. Options for the project itself are found under the Project menu on the Project Options dialog box. Project options include the ability to specify which HTML page serves as your project's start page during testing and which files will serve as the base classes in your project.

You can also set options for your development environment. Options for the development environment are found under the Tools menu on the Development Options dialog box. Examples of development options include which external code editor to use with iPlanet Application Builder, if any, and which web browser to use for viewing HTML pages.

In addition, iPlanet Application Builder also allows you to set runtime options, which affect the runtime characteristics of the iPlanet Application Server used to run your application. Runtime options are found under the Tools menu on the Runtime Options dialog box. Typically, you do not need to change these settings often. Changes may be necessary, for example, if any IP addresses change or if you need to relocate files to a different directory.

Setting Project Options

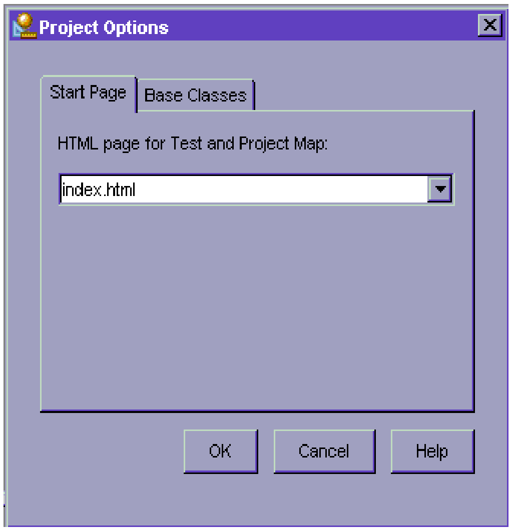
iPlanet Application Builder allows you to perform the following tasks by pulling down the Project menu, then choosing Project Options:

- Setting the Start Page
- Setting Base Classes

Setting the Start Page

iPlanet Application Builder allows you to test a project before deploying it. Each time you successfully test your project, iPlanet Application Builder launches your application from a specified start page. If your project includes an `index.html` page, it is automatically designated the start page until you specify otherwise. You may want to specify a start page other than `index.html` if, for example, you want to test only a portion of your project. The page you specify applies only to testing. For more information, see “Testing Applications” in the chapter titled “Compiling, Testing, and Debugging Applications.”

To set the start page, click the Start Page tab, then choose an HTML page as shown in the following figure:



HTML page for Test and Project Map:

index.html

Setting Base Classes

The Base Classes tab enables you to specify files other than those iPlanet Application Builder creates by default to serve as the base classes of your application.

A base servlet class enables you to reduce redundant coding by centralizing application functionality and letting other servlets inherit from it.

You may want to specify several different Java files to serve as base servlets in your project if, for example, multiple members of a team are simultaneously working on a iPlanet Application Builder project.

To specify which file serves as the base servlet class for your project, click the Base Classes tab, then choose a file from the Base Servlets list, which identifies the classes that you can use as a base servlet for your project.

Setting Development Options

You can establish customized settings for the following features:

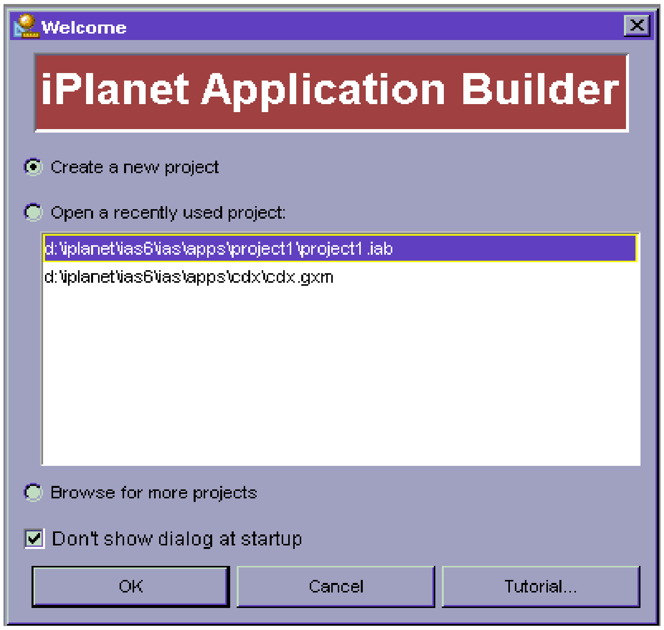
- Startup behavior for iPlanet Application Builder
- Standard or enhanced code generation

- Java compilation
- Text coloring
- HTML editing and image editing
- Source control

You can customize these settings by pulling down the Tools menu, then choosing Development Options.

Setting Startup Options

The Startup tab enables you to specify how iPlanet Application Builder behaves when it is launched. This tab is particularly useful if, upon opening iPlanet Application Builder, you select the "Don't show dialog at startup" checkbox on the Welcome dialog box:

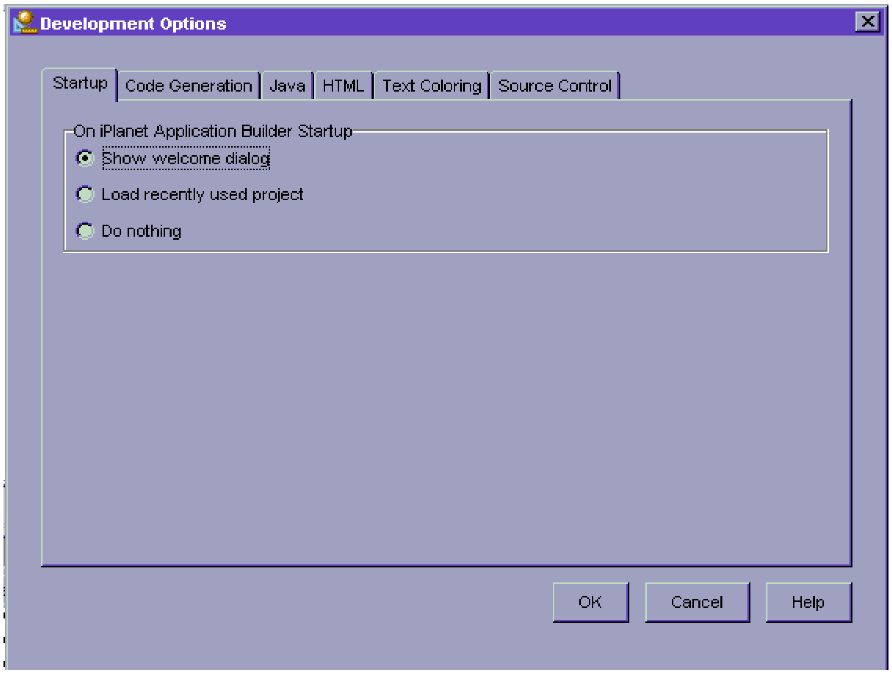


● Create a new project
○ Open a recently used project:

d:\iplanet\ias6\ias\apps\project1\project1.iab
d:\iplanet\ias6\ias\apps\cdx\cdx.gxm

○ Browse for more projects
[✓] Don't show dialog at startup

If you want the Welcome dialog box to appear next time you start iPlanet Application Builder, click the Startup tab in Development Options, then select the appropriate option button as shown in the following figure:



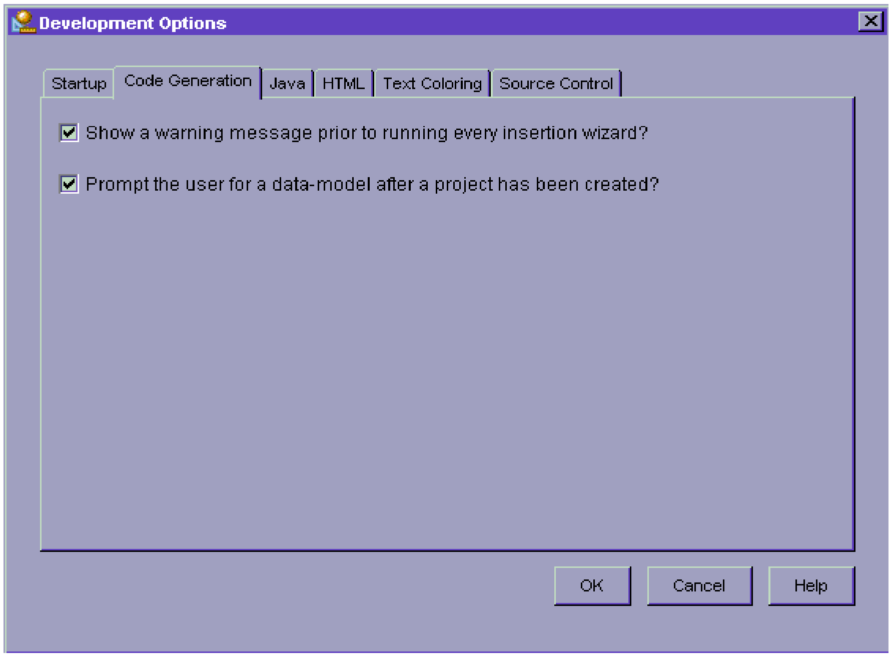
On iPlanet Application Builder Startup

- [✓ Show welcome dialog]
- Load recently used project
- Do nothing

If you choose “Don’t show dialog at startup,” iPlanet Application Builder changes the Startup development option to “Load most recently used project,” which automatically loads the last project. If you do not want either option, you can choose “Do nothing.”

Setting Code Generation Options

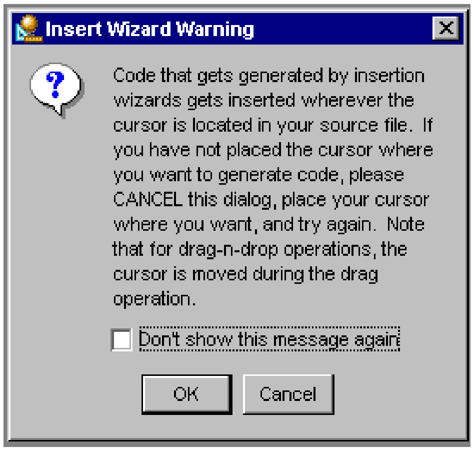
The Code Generation tab allows you to specify the kind of code that iAB generates and to specify other options related to code generation. To set Code Generation options, click the Code Generation tab as shown in the following figure:



☑ Show a warning message prior to running every insertion wizard?

☑ Prompt the user for a data-model after a project has been created?

Check “Show a warning message prior to every running insertion wizard?” if you want to be reminded to set the cursor where code will be generated. If checked, the following message will appear each time you insert code using a wizard:

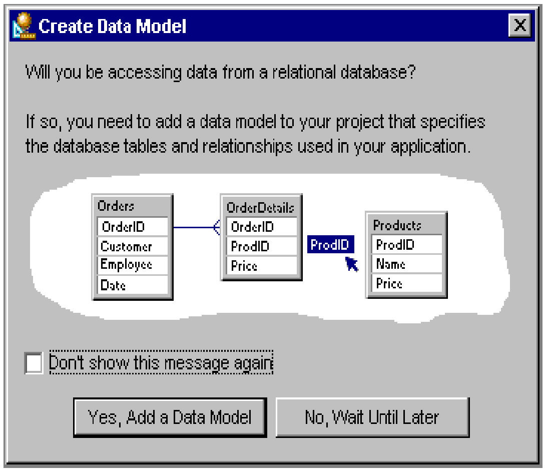


Code that gets generated by insertion wizards gets inserted wherever the cursor is located in your source file. If you have not placed the cursor where you want to generate code, please CANCEL this dialog, place your cursor where you want, and try again. Note that for drag-n-drop operations, the cursor is moved during the drag operation.

☐ Don't show this message again

You can uncheck the Code Generation option or check the “Don’t show this message again” box in the warning dialog box itself, which also unchecks the Code Generation option.

Check “Prompt the user for a data model after a project has been created?” if you want to be reminded to create a data model whenever you create a project. If checked, the following message will appear each time you choose New Project from the File menu:


Create Data Model

Will you be accessing data from a relational database?

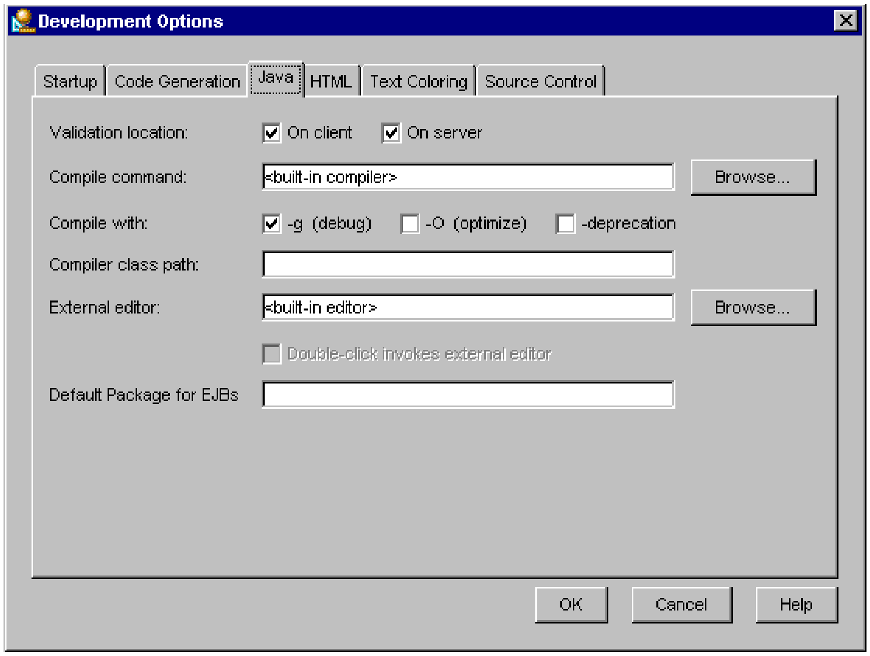
If so, you need to add a data model to your project that specifies the database tables and relationships used in your application.

☐ Don't show this message again

You can uncheck the Code Generation option or check the “Don’t show this message again” box in the warning dialog box itself, which also unchecks the Code Generation option.

Setting Java Compiler Options

The Java tab allows you to customize the behavior of the Java compiler. For instance, you may specify an external compiler and set compiler options. To set Java compiler options, click the Java tab as shown in the following figure:



Validation location: [✓ On client] [✓ On server]

Compile command: <built-in compiler>

Compile with: [✓ -g (debug)] -O (optimize) -deprecation

Compiler class path:

External editor: <built-in editor>

Double-click invokes external editor

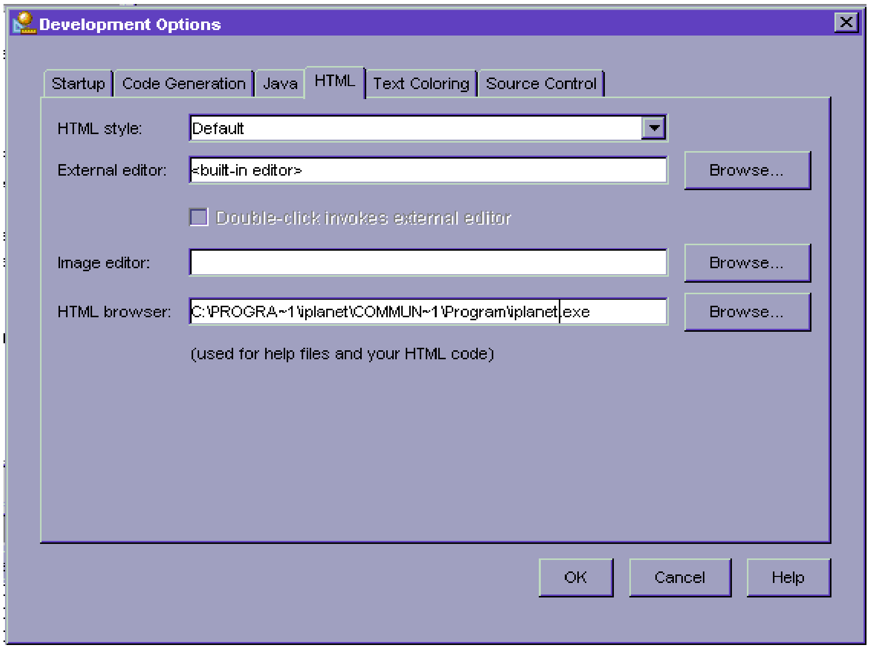
Default Package for EJBs

If you specify an external compiler on the Java tab, iPlanet Application Builder uses the specified compiler rather than the built-in compiler.

If you specify an external Java editor, but do not select the "Double-click invokes external editor" checkbox, iPlanet Application Builder continues to launch its built-in editor. You can opt to open Java files with your external editor by choosing the Open With option found under the Edit menu or by enabling the Double-click Invokes External Editor checkbox on the Java tab.

Setting HTML Options

The HTML tab enables you to specify which HTML style to use in your project pages and JavaServer Pages (JSPs). You can also specify which external editor to use to create HTML pages, which image editor to use to modify images, and which HTML browser to use to view HTML pages and JSPs. To set HTML options, click the HTML tab:



HTML style: Default

External editor: <built-in editor>

☐ Double-click invokes external editor

Image editor:

HTML browser: C:\PROGRA~1\iplanet\COMMUN~1\Program\iplanet.exe

(used for help files and your HTML code)

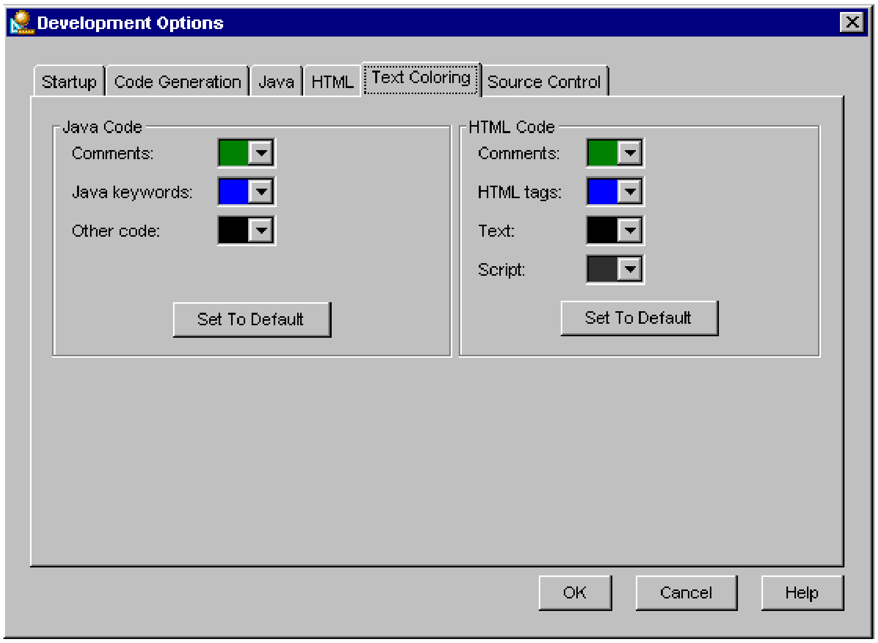
Each HTML style has its own set of colors and font styles. The HTML style you choose defines the appearance of the HTML pages and JSPs in your project. For more information about HTML pages and JavaServer Pages, see in “Creating Presentation Layout.”

If you specify an external image editor on the HTML tab, iPlanet Application Builder launches that application each time you open an image file. If you specify an external HTML editor, but do not select the “Double-click invokes external editor” checkbox, iPlanet Application Builder continues to launch its built-in editor. You can opt to open HTML files with your external editor by choosing the Open With option found under the Edit menu or by enabling the Double-click Invokes External Editor checkbox on the HTML tab.

Setting Text Colors

The Text Coloring tab enables you to customize the color of the code appearing in an editor window. For instance, you may specify that Java keywords within your code appear in red text. These options apply only to editors built in to iPlanet Application Builder.

To set text coloring options, click the Text Coloring tab to display the following dialog box:



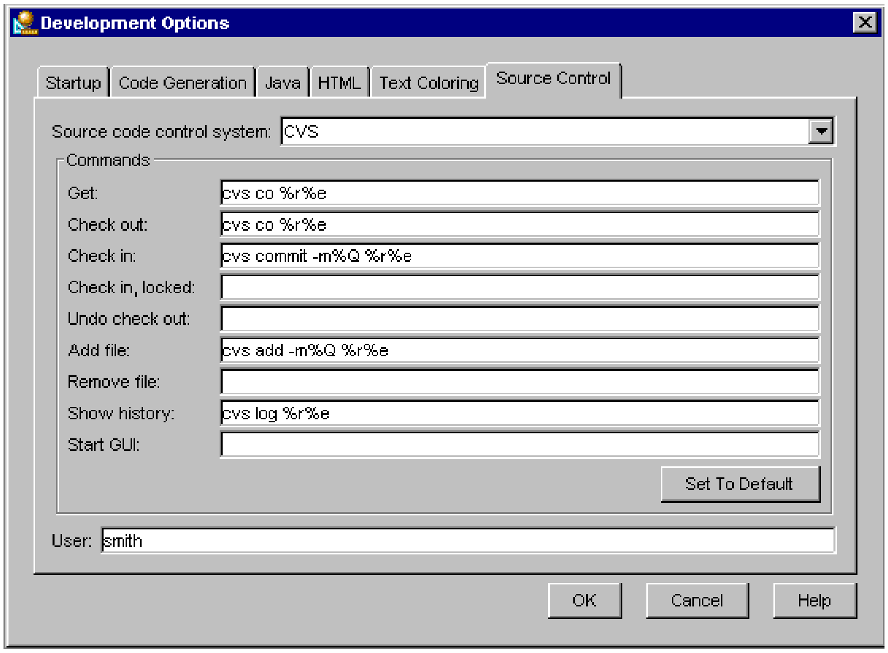
Java Code
- Comments: 🟩
- Java keywords: 🟦
- Other code: ⬛

HTML Code
- Comments: 🟩
- HTML tags: 🟦
- Text: ⬛
- Script: ⬛

Click Help for detailed information about each text coloring option.

Setting Source Control Options

The Source Control tab enables you to specify the external source control program and associated commands. For detailed information on source code control, see "Working with Source Control." To set source control options, click the Source Control tab.



Source code control system: CVS

Commands

Get:	cvs co %r%e
Check out:	cvs co %r%e
Check in:	cvs commit -m%Q %r%e
Check in, locked:	
Undo check out:	
Add file:	cvs add -m%Q %r%e
Remove file:	
Show history:	cvs log %r%e
Start GUI:	

User: smith

You can choose your Source code control system from the drop-down list and make changes to the commands. If you click on Set To Default, the default values you specified to commands will be replaced by the original values that were set when you installed iPlanet Application Builder.

Setting Runtime Options

iPlanet Application Builder enables you to perform the following tasks associated with the runtime characteristics of the iPlanet Application Server used to run your application:

- Changing the Product Key
- Changing File Pathnames
- Setting Options for Message Logging
- Setting Server IDs
- Configuring Database Drivers
- Configuring Data Sources

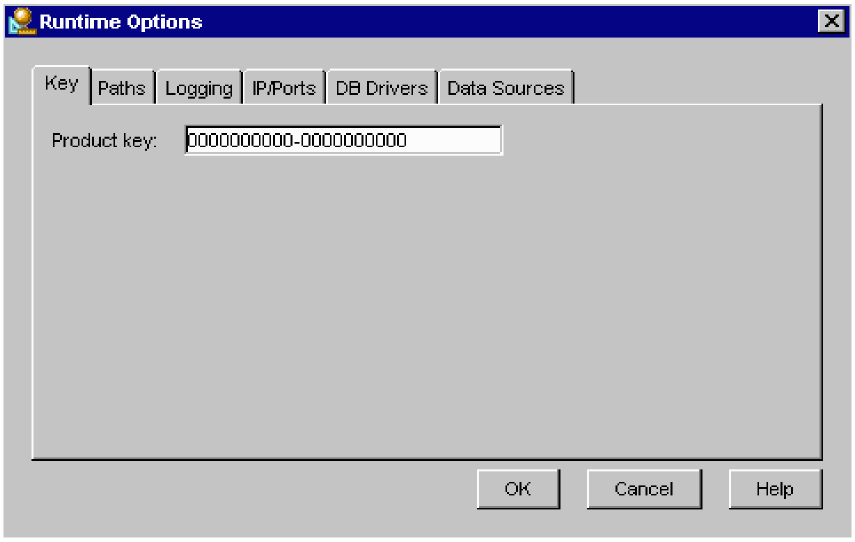
You can perform these tasks by pulling down the Tools menu, then choosing Runtime Options.

Changing the Product Key

Every purchase of iPlanet Application Builder comes with a unique license number called a product key. If you are evaluating iPlanet Application Builder for purchase, you use an **evaluation key**—a temporary product key that expires after a certain time.

After the evaluation key expires, you can no longer start the runtime server. To prevent this from happening, you must purchase iPlanet Application Builder and enter the new, nonexpiring **product key**.

Instead of re-installing iPlanet Application Builder and supplying the new product key during installation, you can change the product key in your current installation. To do so, pull down the Tools menu. Choose Runtime Options and enter the product key in the Product Key field.

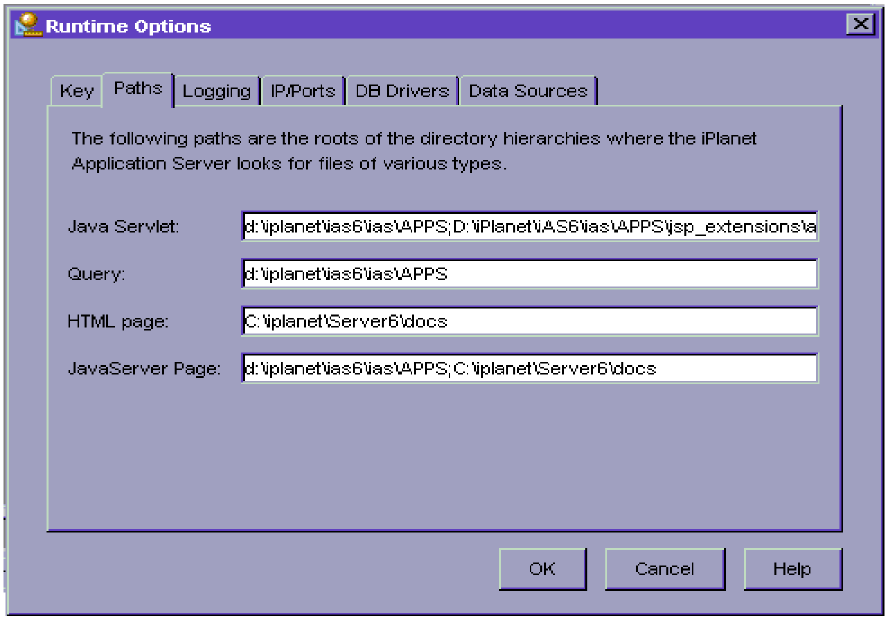


Product key: 0000000000-0000000000

Changing File Pathnames

When a web application is called on to display a page or to execute code, the application looks in particular directories for the files it needs. These file path names are set by the user who installs iPlanet Application Builder. However, if the file system has been reorganized, you may need to change the path names.

To change file path names, click the Paths tab to display the following dialog box:



The following paths are the roots of the directory hierarchies where the iPlanet Application Server looks for files of various types.

Java Servlet: d:\iplanet\ias6\ias\APPS;D:\iPlanet\iAS6\ias\APPS\jsp_extensions\a

Query: d:\iplanet\ias6\ias\APPS

HTML page: C:\iplanet\Server6\docs

JavaServer Page: d:\iplanet\ias6\ias\APPS;C:\iplanet\Server6\docs

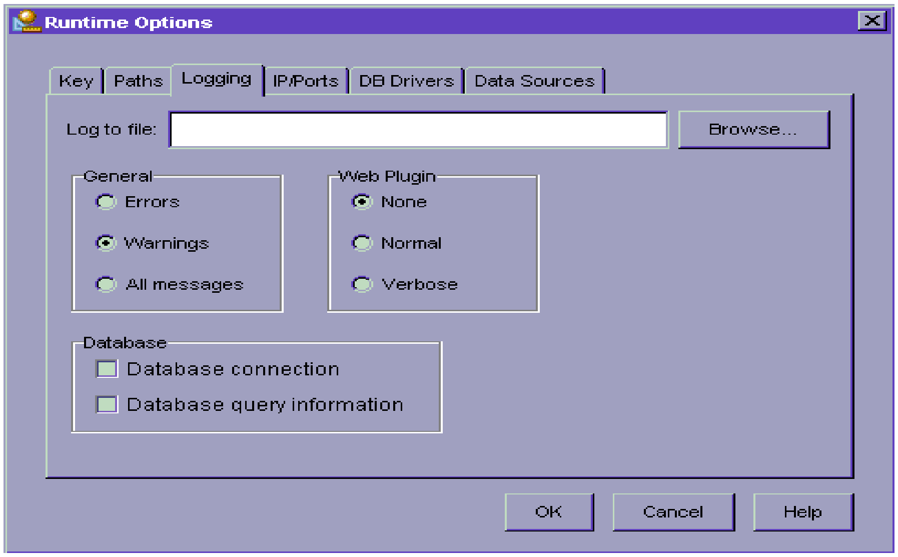
On this tab, you may replace or append existing text. If you specify multiple path names in one text field, directories are searched in order, starting from the left. Use a semicolon as a separator between path names.

To enable the application server to access the HTML directory for its JSP engine, you should include the HTML page directory path name in the JSP text field.

Setting Options for Message Logging

iPlanet Application Server logs the messages generated by application-level and system-level services. These messages describe events that occur while a service is running. You may want to change the amount and type of information that the iPlanet Application Server logs. For example, you can increase the amount of message logging to help diagnose problems; increased logging provides more information about errors and warnings. On the other hand, you can decrease the amount of message logging to use less disk space and increase server speed.

To change the amount and type of information logged, click the Logging Tab:



Log to file:

General: Errors [✓ Warnings] All messages

Web Plugin: [✓ None] Normal Verbose

Database:
- ☐ Database connection
- ☐ Database query information

The Logging tab in iPlanet Application Builder allows you to specify the names of the log files and preferences for the kind and amount of information written to these files.

If you specify a file name for Log To File, a log file is created for the Java server port and the kxs server port. You can determine the port numbers associated with these ports by clicking on the IP / Ports tab. iPlanet Application Builder appends the name of the port to the log file name. For example, if you specify `iabr.log` as the filename and your ports are 10818 and 10819, respectively, iPlanet Application Builder creates log files with the names `iabr.log.10818` and `iabr.log.10819`.

If you do not specify an absolute path, the files relative to the directory in which you started iAB; for example, if you start iAB from the desktop, the directory is `<iab_root>\iab60\bin`. If you specify an absolute path, all directories in the path must already exist; they are not created for you. If you do not specify a file name for Log To File, these files are not created.

In addition to the iAB logs, iAS provides additional logs that are manipulated from the iAS Administrator. For more information, see the *iAS Administration and Deployment Guide*.

The Logging types specify the information you want to log. Logging types are described briefly in the following table:

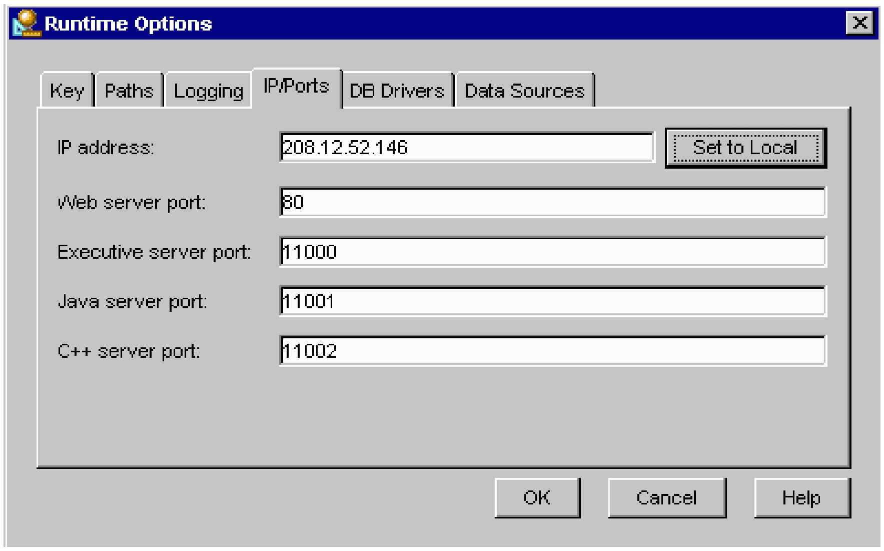
Table 2-1 Logging Types

Type of Logging	Description
General	Logs error messages and warnings. Errors are messages describing a critical failure of a service from which recovery is not likely. Warnings are messages describing non-critical problems that might indicate larger problems. If you select the All Messages button, both Error and Warning messages are logged.
Web Plug-in	Logs requests that come into the web server. You can determine the amount of detail logged by choosing Verbose (more detailed) or Normal (less detailed). Choose None to disable web plug-in logging.
Database	Logs information related to your connection to and interaction with a database. You can choose to record both connection information and query information, or disable both database logging options.

Setting Server IDs

When you install iPlanet Application Builder, you either supply addresses or accept default values for the following in the runtime environment:

- IP address of your machine
- web server port
- executive server port
- Java server port
- C++ server port



IP address: 208.12.52.146

Web server port: 80

Executive server port: 11000

Java server port: 11001

C++ server port: 11002

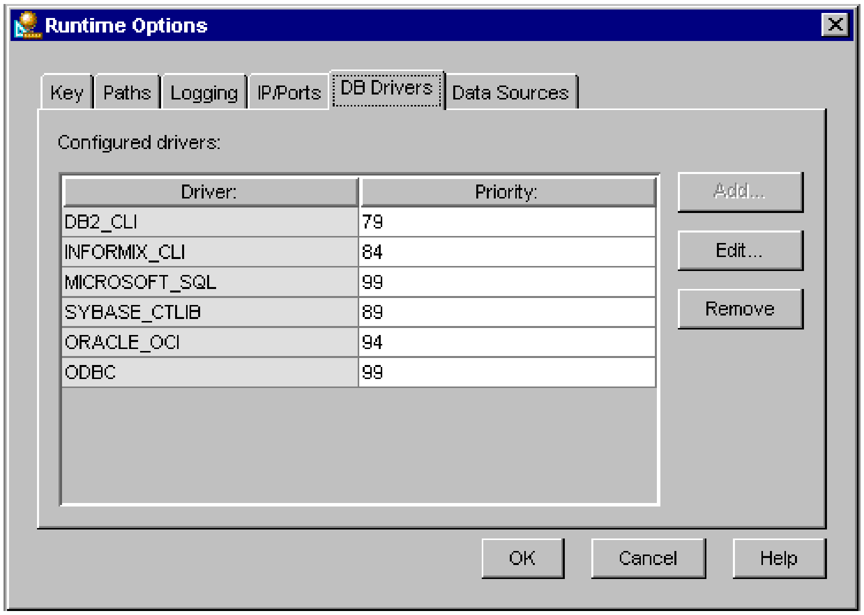
Three of these values, Executive server port, Javaserver port, and C++ server port, correspond to iPlanet Application Server's internal servers. The remaining values, web server port and IP address, correspond to the web server you use with iPlanet Application Builder and the IP address of your local machine, respectively.

If any of these values change, you must update iPlanet Application Builder to match the new values. Instead of re-installing iPlanet Application Builder each time a setting changes, you can change server IDs by clicking the IP / Ports tab as shown in the previous figure. The values are not necessarily dependent on each other; in other words, you may change one value without changing other values.

If you want to update your IP address, click Set to Local; your machine's current address automatically appears in the IP address field.

Configuring Database Drivers

When you install iPlanet Application Builder, you specify the drivers that enable connectivity to a backend database. If you need to add, remove, or change database connectivity after installing iPlanet Application Builder, click the DB Drivers tab as shown in the following figure:



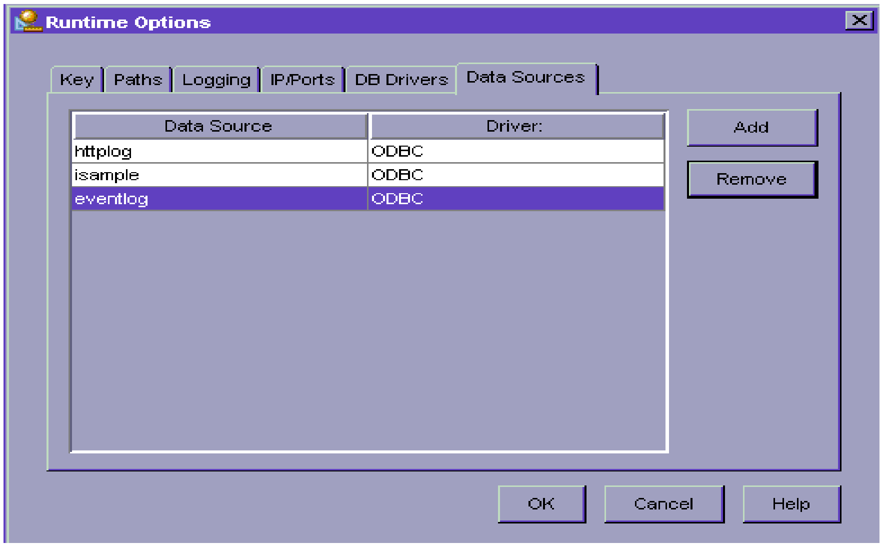
Configured drivers:

Driver:	Priority:
DB2_CLI	79
INFORMIX_CLI	84
MICROSOFT_SQL	99
SYBASE_CTLIB	89
ORACLE_OCI	94
ODBC	99

You can use the Add, Edit, or Remove buttons as needed to modify database connectivity settings. Click Help for detailed information about each option.

Configuring Data Sources

A data source is a logical definition of a backend connection. Whenever you need to add, remove, or modify your database connectivity, you must configure the data sources. To do so, click the Data Sources tab as shown in the following figure:



Data Source	Driver:
httplog	ODBC
isample	ODBC
eventlog	ODBC

You can add or remove a data source as needed. To edit the name of a data source, you can type directly in the Data Source text field. To change driver information, use the Driver drop-down list.

Customizing the Palette

The palette is made up of **palette pages** that can be accessed by tabs at the bottom of the window. The HTML, Form, JSP, and RDBM tabs on the palette are locked, which means that you cannot delete them or remove any components from them. You can create new, unlocked pages using the methods described in the following table.

Table 2-2 Unlocked Page Methods

Customization	Action to Take
Create a new palette page	Right-click the palette and choose New Page. Name the new page.
Delete a palette page	Right-click the palette and choose Delete Page. Confirm the page deletion.
Reorder pages	Left-drag tabs to the desired location.
Reordering components on a page	Left-dag components to the desired order.

Table 2-2 Unlocked Page Methods **(Continued)**

Customization	Action to Take
Moving components between palette pages	Left-drag components to the tab of an unlocked page to move the component to that page. Alternatively, right-click the component and choose Cut, select the page where you want to copy the component, and then right-click the page and choose Paste. You cannot copy components to locked pages but you can copy from them.
Copying components between palette pages	Ctrl+drag components with the left mouse button to the tab of an unlocked page to copy the component to that page. Alternatively, right-click the component and choose Copy, select the page where you want to copy the component, and then right-click the page and choose Paste. You cannot move components to or from locked pages but you can copy from them.
Renaming palette pages	Click the page you wish to rename with the right mouse button and choose Properties. Rename the page. You can rename locked pages.
Viewing and editing component properties	Right-click the component and choose Properties. Component properties can be changed on locked pages.
Deleting components	Right-click the component you wish to delete and choose Delete Item. Confirm the component deletion.

Chapter 3

Creating and Managing Projects

This chapter describes the process of creating and managing projects.

This chapter presents the following topics:

- The Development Process
- Creating a Project
- Adding or Removing Files
- Editing Files
- Viewing and Editing Project File Properties
- Building a Project
- Deploying the Project
- Testing and Debugging

The Development Process

The following flowchart shows the basic steps for developing an iPlanet Application Server application:

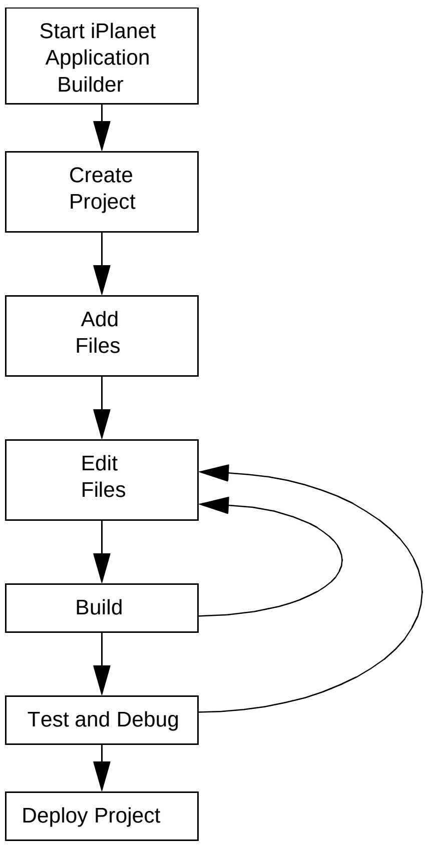


Figure 3-1 Development of an iPlanet Application Server

This section summarizes the development process. Separate sections describe each step in more detail, along with a simple example.

1. Start iPlanet Application Builder.

   This step assumes you have completed the preliminary planning and design tasks, as described in “Designing Applications.”.

2. Create a project.

   When you create a project, iPlanet Application Builder generates a minimum set of files, and they serve as the project’s foundation.

3. Add files to the project.

By reviewing the previously established application design, you know what functionality your application needs, and you know how many web pages the application consists of. These considerations determine what kind of files to add, as well as how many. You can add files one at a time, but you can also use wizards to quickly add a set of files associated with a specific application feature, such as a login page.

4. Edit files.

   As you add files to a project, you can edit them by creating content, by inserting or modifying functionality, or by inserting or modifying the links between them. Different types of files appear in different editor windows. During the editing phase, you use the palette with HTML files. You also use the properties window to review or modify parameter-based information.

5. Compile individual files or build the entire project.

   After adding and editing a few files, you may decide to compile the Java files in the project so far. You can compile an individual file, or you can build the project, which compiles all Java files. As indicated in the previous diagram, you may need to return to the editing phase if compilation errors occur.

6. Test and debug individual files or the entire project.

   After you successfully build the project, you can test it on a server to preview runtime behavior. Testing and debugging go hand in hand, because you can run the server in debug mode. As a result of testing and debugging, you may need to continue editing, then build again.

7. Deploy the project.

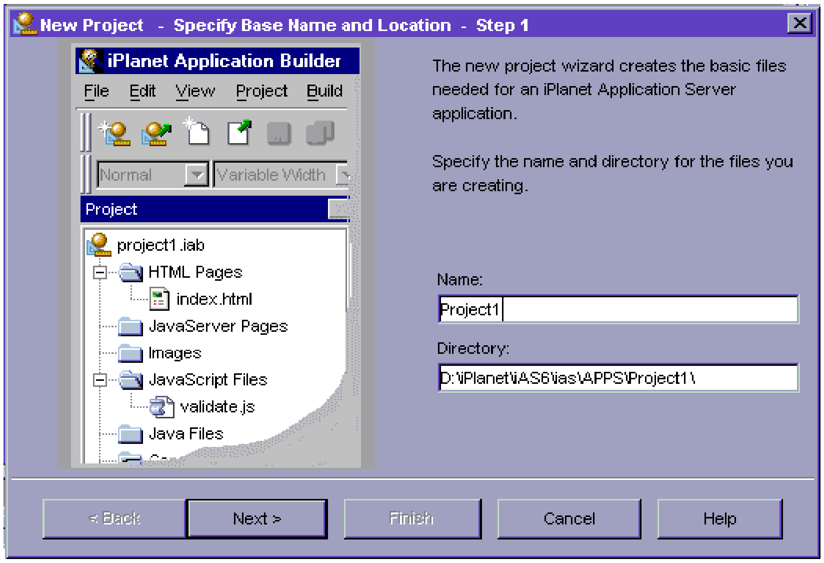
   After successfully testing the project, you deploy it to one or more servers. You have now created a web-based application.

The remaining sections describe each step of the development process in more detail.

Creating a Project

A **project** is a container for related files that constitute an application. Once you define application requirements and design the application flow, you are ready to create a project. First, launch iPlanet Application Builder. Then, to create a project, choose “New Project” either from the Welcome dialog box or from the File menu.

In either case, the new project wizard appears. This wizard guides you through the creation of a simple project, as shown in the following figure:



The new project wizard creates the basic files needed for an iPlanet Application Server application.

Specify the name and directory for the files you are creating.

Name: Project1

Directory: D:\iPlanet\iAS6\ias\APPS\Project1\

The graphic shows the files that the new project wizard can create; the files it actually creates depends on the files you choose to create using the wizard. These files are the foundation of a project. The default files are described in the following table:

Table 3-1 Default Project Files

File Created	**Description**	**Editing Required After File is Created?**
`index.html`	Default start page for the project.	Yes
`validate.js`	Implements JavaScript-based validation for form components. For more information, see “Creating Presentation Layout.”.	No

The wizard also creates a `project.iab` file, which contains information about the files in a project and other information used by the iPlanet Application Builder and the Application Server. This `.iab` file is the file that you choose when you open an existing project.

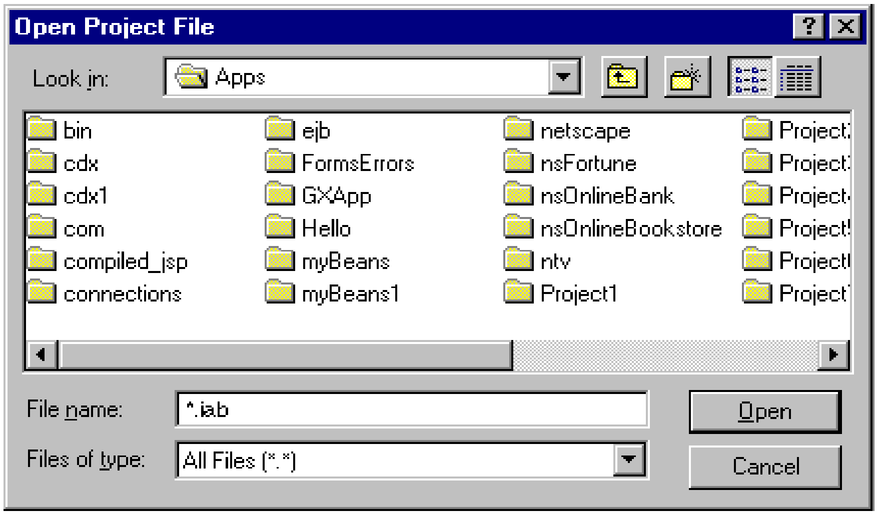
Working with Existing Projects

This section describes how to open projects and either save them or close them.

Opening a Project

Opening a project means opening its associated `.iab` file. You can open a project by using the File menu to choose Open Project, by clicking the Open Project button from the standard toolbar, or by double-clicking one of the files listed in the Welcome dialog box. When you open a file from the Welcome dialog box, you also can select “Browse for more projects” to search the file system for another `.iab` file.

When you search the file system, a dialog box appears:



To open the Sample project, you double-click the Sample folder to list its `.iab` file. After selecting this file, you can double-click on it or click Open.

Only one project can be open at a time. If you try to open a second project when a project is already open, the first project closes after prompting you to save any changed files.

Saving and Closing Projects

Saving a project means saving all files that compose a project. To save a project, pull down the File menu and choose Save All or click on the double-disk icon in the toolbar. To close a project, pull down the File menu and choose Close Project.

When you save a project, the project state is automatically registered with the local application server.

Adding or Removing Files

You can add files to a project in any of the following ways:

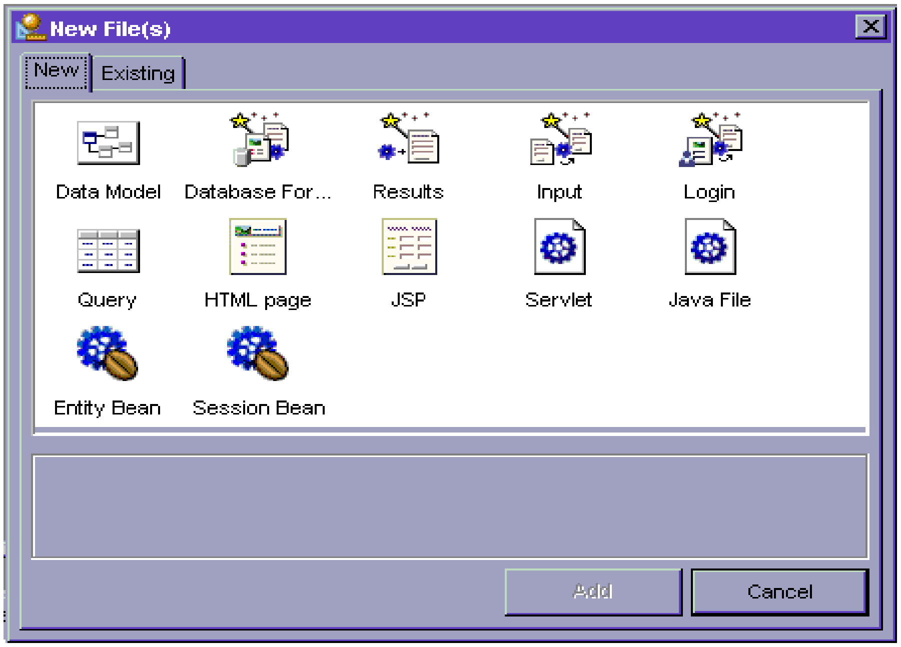
- Create files one at a time, from scratch.
- Create multiple files, using wizards.
- Add existing files.

You can also remove files that are not needed in the project.

Whenever you create files in iPlanet Application Builder, they are automatically added to the current project. This means the files appear in the project tree and are tracked by the `.iab` file. By contrast, previously existing files do not become part of a project unless you specifically add them.

Creating Files

To create files, you can either pull down the Project menu and choose Add Files, or you can pull down the File menu and choose New and then choose Files. In either case, the New Files dialog box appears:



To create files, either double-click an icon, or select an icon and click Add. You can use the New Files dialog box to create the following individual files:

Table 3-2 Individual Files created using the New Files dialog

File Type	Description	Where Added to
HTML page	A static HTML file.	Project's subdirectory in the web server's document directory.
Servlet	An instance of a Java class that implements the presentation logic of an application.	Project's subdirectory in the iAS applications directory.
Java file	A generic helper class. It starts out as a blank file, with no generated code. Therefore, you can use a Java class to create Java source files from scratch.	Project's subdirectory in the iAS applications directory.
Data model	Specifies the database tables and relationships to be used in the application.	Project's subdirectory in the iAS applications directory.
Query	Lets you define SQL queries. A query can be called from a servlet or a JSP.	Project's subdirectory in the iAS applications directory.

Table 3-2 Individual Files created using the New Files dialog *(Continued)*

File Type	Description	Where Added to
JavaServer Page (JSP)	An HTML file, with Sun-specified extensions, that is executed on the web or application server instead of on the client server, dynamically creating pure HTML, which is then sent to the client.	Project's subdirectory in the iAS applications directory.

You can also use the following wizards to create various sets of files:

Table 3-3 Wizards

Wizard	Description
Database Forms	Lets you create master/detail forms and reports, with an optional search page. This wizard creates a page containing a data query form; query files and data models to establish interaction with a relational database; and servlets to process and display the query results.
Results	Creates a servlet and a JSP to display the results of a query.
Input	Creates three files: an HTML page containing a data input form, a servlet to process the form, and a JSP to display the results. Can also produce a query if needed.
Login	Creates files that perform the login function: a static HTML page containing a login form, a servlet to process the form, and a JSP to display results.
Session Bean	Creates a session EJB's source files, including the remote interface, home interface, enterprise bean class, and an `.ebx` file.
Entity Bean	Creates an entity EJB's source files, including the remote interface, home interface, enterprise bean class, and an `.ebx` file.

Adding Existing Files

Files that were created outside of iPlanet Application Builder are not automatically tracked by a project's `.iab` file (and are therefore not considered part of a project) unless you add these files explicitly using iPlanet Application Builder. The files you add must exist within the iAS applications directory tree or in the web server's document directory.

To add files that were created outside of iPlanet Application Builder, perform the following steps:

1. Using your operating system, copy the files to the appropriate directory:

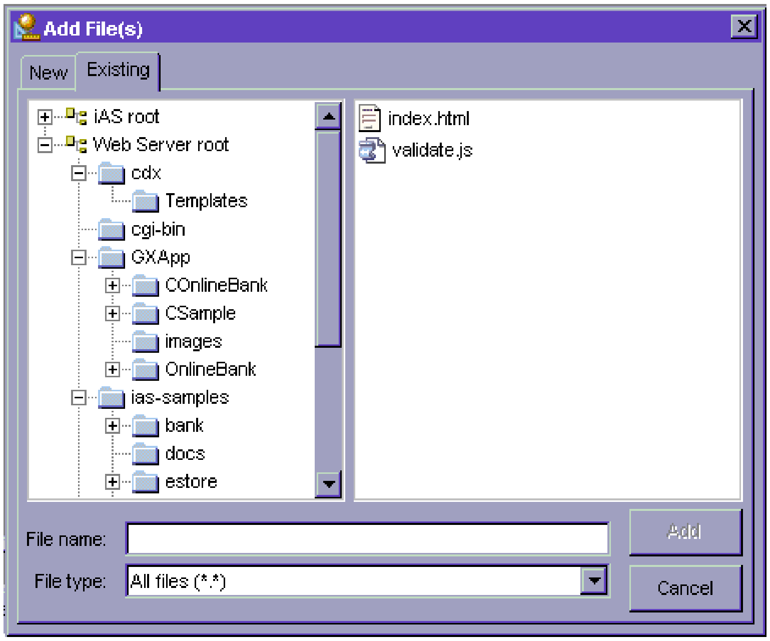
   ❍ Copy HTML pages (static files), JavaScript files, and images to your web server's document directory.

   ❍ Copy other files (queries, data models, Java source, Java Server Pages) to your project root.

   Be sure to copy any files referenced by the file you are copying.

2. In iPlanet Application Builder, display the Add Files dialog box by pulling down the Project menu and choosing Add Files. You can also choose New then Files from the File menu and select the Existing tab.

3. Locate and select one or more files to add to the project.



4. Double click to add a file or click Add.

   The file and any files it references are added to the project.

Removing Files from a Project

When you remove a file from a project, you delete the reference to it in the `.iab` file. The removed file no longer appears in the Project window. By default, removing a file from a project does not delete it from the file system.

NOTE Do not to remove files that other files depend on.

To remove a file from the project, perform the following steps:

1. Select the file in the Project window.
2. From the Project menu, choose Remove *File*. For example, to remove `index.html`, choose “Remove index.html”. You may also right-click on the file and use the “Remove” command.

   A confirmation dialog box appears.
3. Select the checkbox if you want to remove the file from the file system.
4. Click Yes to remove the file from the project.

Editing Files

After you add files to a project, you edit them to fill in the details. You can edit the following types of files within the iPlanet Application Builder environment:

- HTML files
- Java and JavaScript files
- JavaServer Pages
- data models
- query files

You can optionally specify external editors for HTML and Java files. In addition, you can edit image files by designating a third-party image editor to do so. For information on setting external editors, see “Setting Application Builder Options.”

You can open a file for editing in any of the following ways:

- Double-click the file in the Project window or project map.
- Right-click the file in the Project window or project map, then choose Open.

- From the File menu, choose Open File.
- From the Edit menu, choose Open Selected File(s).

The type of file you open determines which editor window appears. When a file is open, you can perform standard editing tasks such as copying, pasting, or searching for text. You execute these tasks using either the Edit menu or the standard toolbar.

Saving, Closing, and Renaming Files

If a file has unsaved changes, its name appears in the Project window with boldface type and with an asterisk (*) appended to the name. Saving a file restores the name to its normal appearance in the Project window.

You can save an individual file or save all files in a project. To do so, choose Save or Save All, using either the File menu or the standard toolbar.

To close a file, use the File menu or click the Close button in the editor window. If there are unsaved changes, you are given the option of saving the file first.

To rename a file, choose Save As from the File menu, and enter the new file name.

Viewing and Editing Project File Properties

This section describes viewing and editing project file **properties**. Properties determine the appearance and behavior of objects in a project.

Each object in a project, including HTML and Java files as well as components and structural objects in those files, has a set of properties associated with it. Each property consists of a name, a description, and a value. The value of a property is usually an editable variable that affects the appearance or behavior of the object to which the property belongs.

The following topics are described in this section:

- Viewing Properties
- Editing Properties
- Using Custom Property Editors

Viewing Properties

View and edit properties by using the **Properties window**.

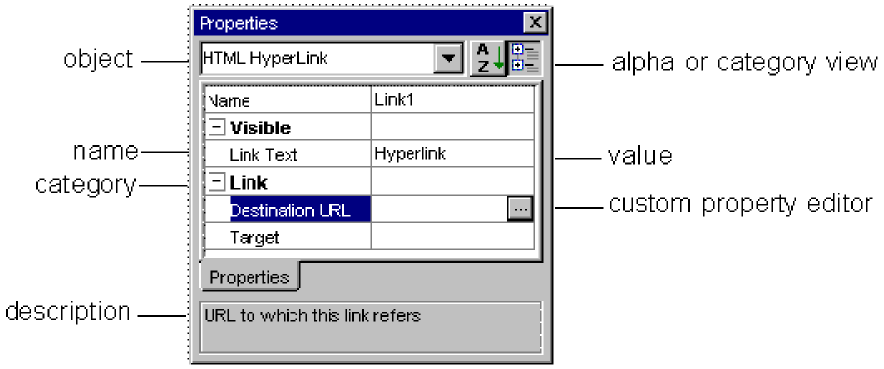
The object whose properties are being displayed appears in the top drop-down box. Property names appear in the left column, and their corresponding values appear in the right column. When a property is selected, its description appears in the description box at the bottom of the window.

Some properties have special features to help you edit their values. For more information, see “Editing Properties”.

Click the alphabetical view button to view properties in alphabetical order.

Click the category view button to view properties in order by category.

The following illustration shows the Properties window in category view:



Name	Link1
⊟ **Visible**	
Link Text	Hyperlink
⊟ **Link**	
Destination URL	...
Target	

URL to which this link refers

You can view the properties of any object in your project.

Editing Properties

The values of many properties are plain text, and can be edited according to the property’s constraints by clicking in the property value and typing an appropriate entry.

For properties that must contain one of a restricted set of values, the set of possible values is represented as a drop-down box . To choose an option, click the down-arrow and select one of the listed options.

Some properties can be edited with a **custom property editor**, a dialog box created for a specific property. These properties show a button ... in the property definition box when you edit the definition. This button launches a custom property editor for that property, which helps you specify various attributes for this property value. For example, click the custom property editor button for the `queryName` property on a Loop component would help you set the `ResultSet` and the format for that property. For more information on custom property editors, see "Using Custom Property Editors".

Using Custom Property Editors

Custom property editors are specialized dialog boxes that help you provide correct information to complex properties. Custom property editors exist which can help you with the following tasks:

- Editing Data-Bound Properties
- Editing URLs
- Editing List Items
- Editing Colors

Editing Data-Bound Properties

There are three standard **data-bound properties** that together make up a data-bound object, linking a single column of data to a component. These properties are: a **RowSet**, which identifies the source of the data (for example, a query); a **RowField**, which identifies the specific column to display (for example, a field in a query); and, optionally, a **Display Format**, which determines a format for the data if needed.

A **data expression** is one or more combinations of a RowSet with a data field, optionally with a display format and / or static characters. You can use a custom data expression to provide multiple combinations of RowSet, RowField, and Display Format inside one component.

For more information on these properties, see "Creating Presentation Layout."

Editing Formats

A **display format** allows you to format dates or numerical data at run time. This feature is also called a **format mask**, as it masks the raw data with a different format for viewing.

To use a custom property editor to edit the Display Format property, click the custom property editor button ... in the property definition.

Choose one of the following options. When you select a format from the drop-down list, the format string and an example showing runtime appearance appear below. For more information on formatting, see "Creating Presentation Layout".

Table 3-4 Format Options

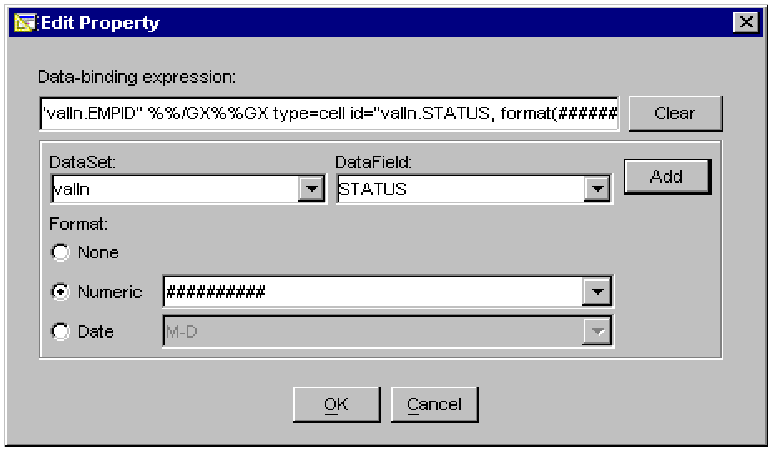
Option	Usage
None	No display format applied. This is the default.
Numeric	Indicates that the value is numeric. In general, # indicates a digit if necessary (no leading zeros) and 0 indicates a digit absolutely (zero-fill if necessary). Most other punctuation is printed as shown. For example, the data `12345.6` formatted with `$###,###.00` yields `$12,345.60`.
Date	Indicates that the value is a date. In general, D indicates a value for the day, M for month, Y for year, h for hour, m for minute, and s for second. Punctuation appears as shown, if needed. The more times the indicator is repeated, the more complex the format (DD yields 04 for the day, while DDDD yields Friday).
Custom	Custom format. Select from the drop-down list, or develop your own.

Output formatting is ignored for `valIn`, `valOut`, and session variables.

Editing Complex Data Expressions

A RowSet/RowField combination yields one column of data from a query. You can provide more than one column of data in a given data-binding, however, by using a custom-built **data expression** that states which RowSets and RowFields appear in the single display.

To use a custom property editor to create a complex data expression, click the custom property editor button ... in the data expression property definition. The custom property editor, shown below, appears:



Data-binding expression: 'valIn.EMPID" %%/GX%%GX type=cell id="valIn.STATUS, format(######

DataSet: valIn

DataField: STATUS

Format: ( ) None [✓ Numeric] ( ) Date

Numeric: ##########

Date: M-D

In general, follow these steps for each RowSet/RowField combination you need in your expression:

1. Choose a RowSet and a RowField.
2. Choose a format mask, if needed.
3. Click Add.

   The expression for that field, represented as `<rdbm:field>`, is added to the “Data-binding expression” box. At runtime, this expression is replaced with dynamic data.

4. Add any text or HTML markup, such as punctuation or spacing, between fields.

   For example, if you add two `RowFields` called `LastName` and `FirstName`, you might want to add a comma and space between them by first adding `LastName`, then adding a comma and a space after the expression, then finally adding `FirstName`. The final product might look something like the following expression (note the comma after the `valIn.LastName` tag):

   ```
   <rdbm:field query=”query” name=”LastName”/>, <rdbm:field
   query=”myquery” name=”FirstName”/>
   ```

5. When you have added all the `<rdbm:field>` tags you require, click OK to insert the data expression into your JavaServer Page.

Editing URLs

There are two types of URLs for developing iPlanet applications:

- A **link URL** specifies the target for a hypertext link.

- An **image URL** specifies the location of an image.

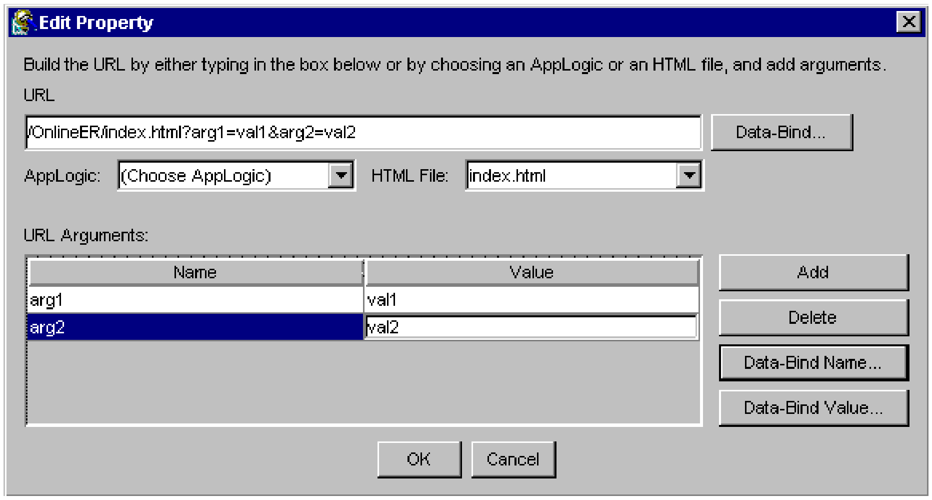
For both link and image URLs, the URL can be relative (local server) or absolute (local or remote server). The URL can be determined dynamically if the component that requires it is in a JSP.

Custom property editors exist for both types of URLs, as described in the following sections.

Editing Link URLs

A link URL is a target for a hypertext link. The target can be an HTML page, a servlet, or any other web object accessible from the user's browser.

1. To specify a link URL, click the custom property editor icon. The following illustration shows the resulting dialog box, with some sample data:



Build the URL by either typing in the box below or by choosing an AppLogic or an HTML file, and add arguments.

URL: /OnlineER/index.html?arg1=val1&arg2=val2

AppLogic: (Choose AppLogic)   HTML File: index.html

URL Arguments:

Name	Value
arg1	val1
arg2	val2

2. You can specify an object in your project by selecting an HTML page or a servlet from the drop-down boxes, or you can enter a URL by hand in the URL text box.

3. To add arguments (name/value pairs linked with = and separated by &), click Add and specify a name and value.

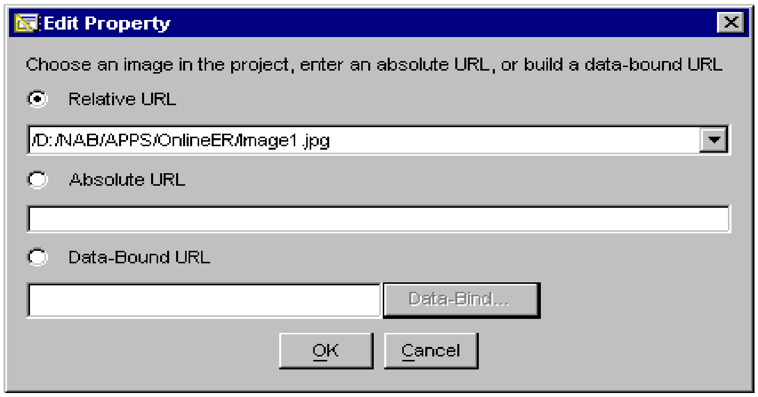
4. If this link is in an HTML page, you can bind any part of the URL to dynamic data from a servlet by clicking the associated Data-Bind button. Create a data-binding expression using the resulting dialog box as described in "Editing Complex Data Expressions".

Editing Image URLs

The source code for an image on an HTML page or template is located by a URL. The URL can be relative to the page, or a fixed, absolute address. If the URL is in a JSP, it cannot be relative to the page; it must be either absolute or relative to the application root. If the image resides on a JSP, the source can be determined dynamically by a data binding.

1. To edit an image URL, click the custom property editor for the Image Source or Low Res. Source property.

   The following dialog box appears:



Choose an image in the project, enter an absolute URL, or build a data-bound URL

(●) Relative URL

/D:/NAB/APPS/OnlineER/Image1.jpg

( ) Absolute URL

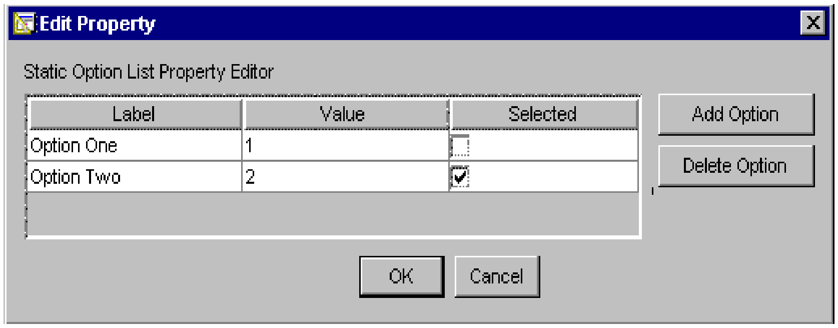
( ) Data-Bound URL

1. To insert a relative URL to an image in your project, choose an image from the drop-down box.
2. To insert an absolute URL, click the Absolute URL radio button and enter the URL.
3. If this image is in an HTML template, you can determine the URL with dynamic data from a servlet by clicking the Data-Bound URL radio button and then clicking the associated Data-Bind button. Create a data-binding expression using the resulting dialog box as described in “Editing Complex Data Expressions”.

Editing List Items

Form list components, such as Drop-Down List, List Box, and Radio Button Group, are often populated by dynamic data in templates. On static pages or in templates where the list options are not dynamically generated, you can edit lists that are not data-bound by selecting the List Items property and clicking the custom property editor icon.

The following dialog box helps you create list items:



Label	Value	Selected
Option One	1	☐
Option Two	2	✓

Set up list items by creating name/value pairs. If you want an item to be selected by default at runtime, click the Selected checkbox for that option. To add or delete an item, click the appropriate button on the right. When you are finished editing list items, click OK, or click Cancel to abandon changes.

The Drop-Down List component creates a drop-down list with an HTML `select` statement with the attribute `size=1`. This implies that only one option can be selected at a time. If you check the Selected checkbox for more than one option, behavior at runtime is undetermined. In iPlanet Navigator 4.*x*, the drop-down box becomes a scrolling list one item high.

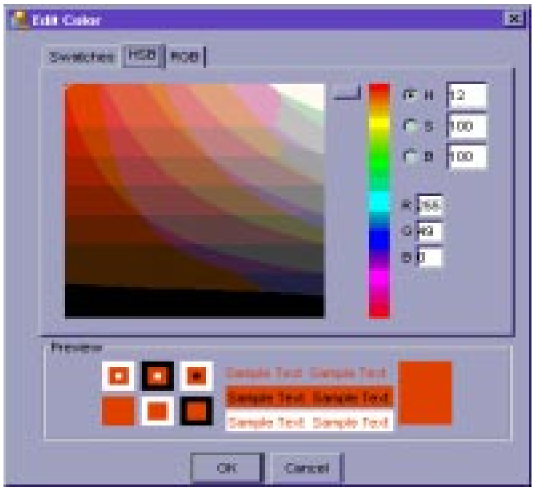
Similarly, to enable multiple values to be initially selected in the List Box component, you must ensure that the Multiple property is set to `true`.

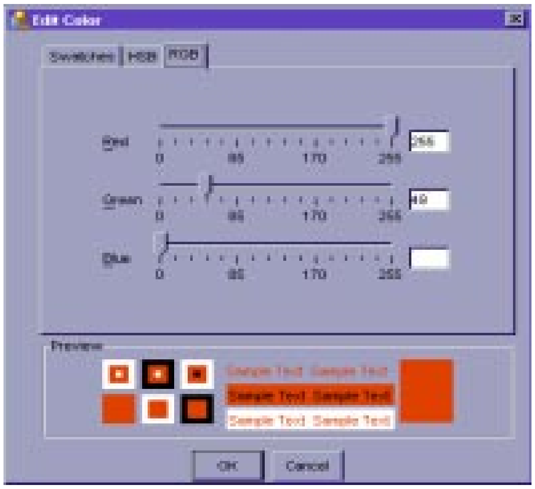
Editing Colors

Some properties indicate a color, which must be specified using either RGB (a hexadecimal number representing a combination of red, green, and blue) or HSB (a number representing hue, saturation, and value).

You can edit color properties by specifying HSB values. The custom color editor helps you choose colors using either method. Click the tab corresponding with the method you prefer. When specify a color in one mode, the other mode is updated. In both tabs, the selected color is shown in the rectangle on the bottom right.

The following illustrations show the color property editor in HSB mode:





Building a Project

The Build menu includes commands for compiling Java files. Shortcuts to these commands appear on the build toolbar. Building a project is comparable to using a `make` utility to compile a set of source files. If building generates errors, check the Messages window to determine which file or files require editing.

Your personal preference determines how often you decide to compile code. For example, with complex code, you might compile one source file every time you make a small change to it. By contrast, you might decide to wait until there are several Java files in the project, then compile all of them using the Build Project command.

Furthermore, you can choose to skip the build commands altogether and proceed to the testing phase, because building happens implicitly whenever you test a file or project.

For more information, see “Compiling, Testing, and Debugging Applications.”

Deploying the Project

When a project is complete, and it runs as expected on the test server, you are ready to deploy it as an application. Deploying the project means copying the files to a live server and registering them with the server. In other words, the application becomes available to end users.

You can deploy an application using the Deployment Tool, a separate tool accessible from the iPlanet Application Server (iAS) Administration Tool, or from the iPlanet Application Builder. When you deploy an application, the Deployment Tool installs all the application files and registers all the components on the destination server.

When you deploy the application `.ear` file using the iAS Deployment Tool, some archived application files are automatically distributed to their appropriate directories on one or more servers and registered with iAS. For example, Java class files, static HTML files, JSP files, and files in your web application module are all automatically sent to the application directory of your application server. EJBs remain and are accessed inside the `.ear` file.

For more information on deploying or downloading applications, see “Deploying Applications.”

Testing and Debugging

You can test an individual file or an entire project. Testing an individual file lets you preview it in a web browser. Similarly, testing a project lets you preview its behavior. In this case, the browser displays the project’s start page.

When you test a file or a project, iPlanet Application Builder first saves and compiles files as needed. If compilation fails, you must edit the affected files, then build or test again. Once the build succeeds, the application server starts, and the test page appears in the browser.

At this point, you can test the file or project for intended behavior. For example, you can click links, click buttons, and enter data into text fields to see whether the project produces the desired results.

If the application does not work as intended, you may need to edit one or more files, or you may need to debug the project to isolate the problem.

For more information, see “Compiling, Testing, and Debugging Applications.”

Chapter 4

Creating Data Access Logic

This chapter describes the concepts and tasks for creating a model of your data source, establishing the specific relationships between those data, and how to use iPlanet Application Builder to create queries that use a data model to access and retrieve information from a data source.

The following topics are described in this chapter:

- About Data Models and Queries
- Creating a Data Model
- Importing a Third-Party Data Model
- Creating Queries
- Editing Generated Code: Queries
- Adding Conditionals to a Query
- Adding Joins to a Query
- Sorting Query Results

About Data Models and Queries

This section provides an overview for data models and queries. For a more complete discussion of how to create and edit data models, see "Creating a Data Model." For more information regarding queries, see "Creating Queries."

About Data Models

A **data model**, indicated by the `.kdm` file extension, is an entity relationship (ER) diagram that specifies the data source tables and relationships used in your application (an **ER diagram** describes the attributes of database entities and the relationships among them). You create your data source queries by first constructing or importing a data model that includes tables and the relationships (**joins**) between them from a specific portion of your data source. You must define a data model before creating query files or pages that access database data.

To build a data model, you select data sources, and then add and / or remove tables, fields, and relationships. Your iPlanet Application Builder queries are based on the information contained within your data model(s). Once you create a data model for a data source, you can reuse that model extensively.

About Data Access Queries

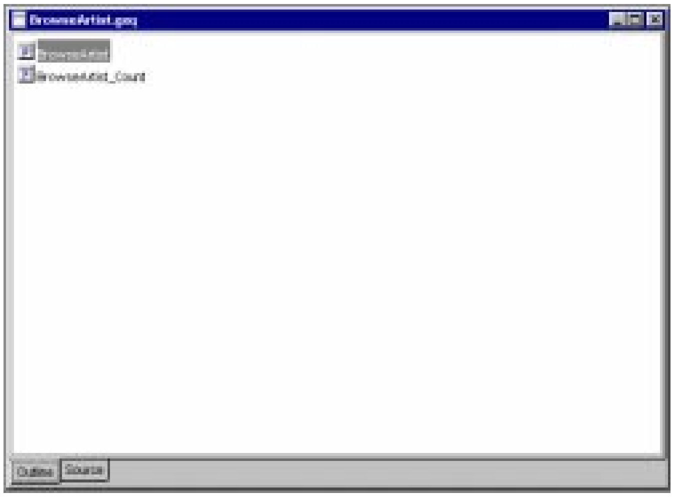
A **query** is a statement that specifies which data to retrieve from, insert into, delete from, or update within a data source. A **data source** is a collection of data electronically stored within a relational database, legacy system, or object database.

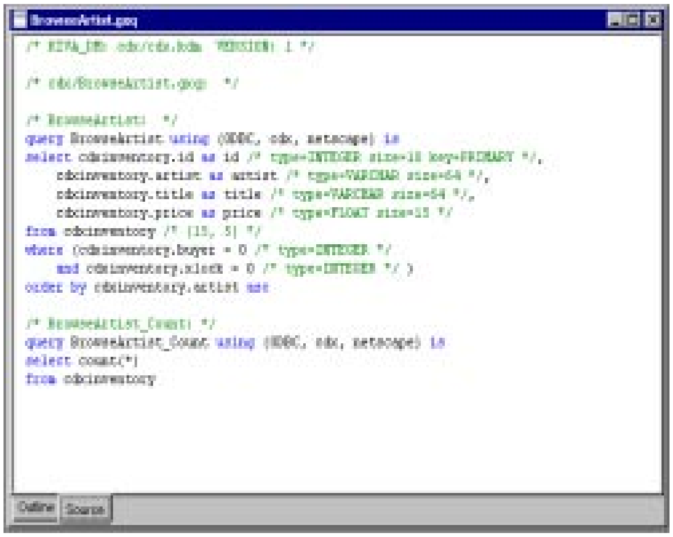
Typically, the results of a query are displayed in a report. iPlanet Application Builder stores queries in files of type `.gxq` that can be subsequently executed by an associated servlet or JavaServer Page (JSP) file (via a JDBC method) running in iPlanet Enterprise Server. For more information, see the *Programmer's Guide.*

The iPlanet Application Builder query window has two view tabs: Outline and Source. The **Outline view** shows each query. The **Source view** shows the actual SQL in the file.

The query command options (via the Insert menu) are enabled along the top portion of the iPlanet Application Builder screen once you open or create a query file. Additionally, you can display the Query toolbar by choosing Toolbars – Query from the View menu.

You edit the SQL code that corresponds to a query directly, either by opening the query and choosing the source tab or double clicking on the query and choosing the SQL tab from the outline view. The following illustrations show the outline and source views for an example query:





```
/* KIVA_DM: cdx/cdx.bdm  VERSION: 1 */

/* cdx/BrowseArtist.gxq  */

/* BrowseArtist:  */
query BrowseArtist using (ODBC, cdx, netscape) is
select cdxinventory.id as id /* type=INTEGER size=10 key=PRIMARY */,
    cdxinventory.artist as artist /* type=VARCHAR size=64 */,
    cdxinventory.title as title /* type=VARCHAR size=64 */,
    cdxinventory.price as price /* type=FLOAT size=15 */
from cdxinventory /* [15, 5] */
where (cdxinventory.buyer = 0 /* type=INTEGER */
    and cdxinventory.stock = 0 /* type=INTEGER */ )
order by cdxinventory.artist asc

/* BrowseArtist_Count: */
query BrowseArtist_Count using (ODBC, cdx, netscape) is
select count(*)
from cdxinventory
```

Using Data Models and Queries

iPlanet Application Builder enables you to design queries based on one or more data models that map the high-level relationships between your data to the actual data themselves. This data model represents the physical data that is stored in the data source.

A data model offers you the following benefits for your project's database query (a file of type `.gxq`) design:

- Provides a reference to the structure of your data source(s).
- Defines the relationships that iPlanet Application Builder uses as the default table joins in editing queries and running various data source wizards.
- Allows your data architect to define how tables relate to each other.
- Provides a means for breaking up a large data source schema into separate multiple subsets, so you don't have to retrieve large amounts of metadata (data about the tables in your data source) from your data source each time you define your queries.

Using iPlanet Application Builder, you can create database queries by importing an existing PowerDesigner data model (a file of type `.pdm`) or by generating a new data model (a file of type `.kdm`).

The following figure shows the various data model inputs you can use for designing a data model and the resulting output file (a file of type `.kdm`). Once you have created a data model, you can begin designing the database queries.

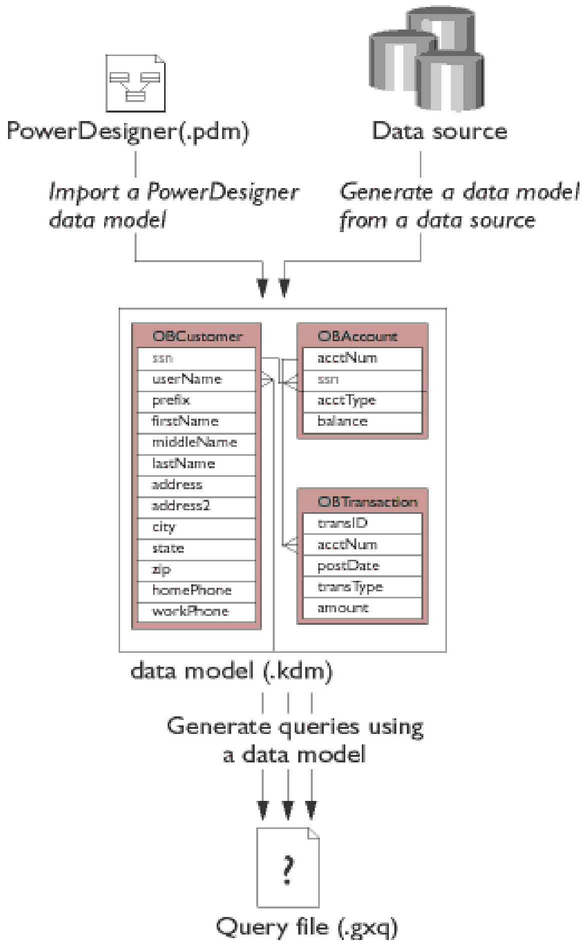


Figure 4-1 Data Model Inputs

You can create one or more data models, depending on the level of organization that you require for your data source queries. For example, you might decide to have separate data models for the accounting, inventory, and sales organizations within your company rather than having one data model for your entire company data source.

Dividing data source tables into multiple data models enables query designers to view a smaller subset of the entire database each time a new query is created. By including only the relevant data source tables required for a particular department, a query designer can design queries faster, since iPlanet Application Builder displays only the relevant tables and fields within that department's data model.

In addition, if your database is particularly large, yet a specific data model represents only a small subset of your data, application performance may be improved, since this multiple data model architecture enables the query designer to avoid searching through the entire database when a new query is only associated with a small subset of the database.

Creating a Data Model

To create a new data model, use iPlanet Application Builder to perform the following tasks:

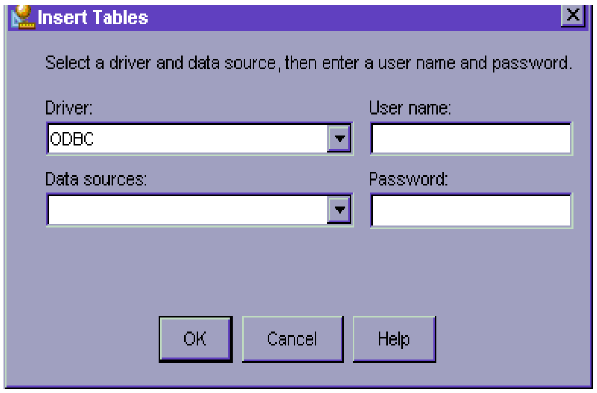
- Selecting a Data Source
- Adding Data Source Tables
- Viewing or Editing Data Source Table Properties
- Viewing a Column's Properties or Changing the Column Alias
- Specifying Relationships Between Data Source Tables
- Adding Calculated Fields

Selecting a Data Source

You use a data source to define the specific database tables you want to use. To select a data source, perform the following steps:

1. Choose New – Data Model from the File menu.

   iPlanet Application Builder prompts you for the type of connection. Once you select the connection type, the following Insert Tables dialog box appears:



Insert Tables

Select a driver and data source, then enter a user name and password.

Driver: ODBC

User name:

Data sources:

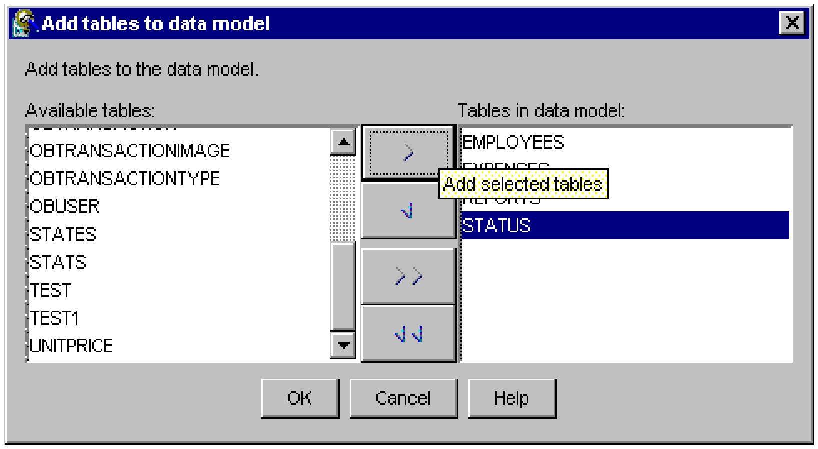
Password:

NOTE The query's connection information must match the data model's connection information; othewise, iPlanet Application Builder considers the query to be a custom SQL query.

- If you selected ODBC as your driver type, select the name of the data source in Data Sources field.

2. Type the user ID required by the selected data source in User Name.
3. Type the user password required by the selected data source in Password.

   iPlanet Application Builder logs you in to the selected data source. If the login is successful, iPlanet Application Builder displays the following Add Tables to Data Model dialog box:



Adding Data Source Tables

You specify data tables for your data model to determine which portion of your data source you want to use to model your data. To use iPlanet Application Builder to add data source tables, perform the following steps:

1. Select one or more table names in Available Tables.

To select multiple tables, you must perform a Shift + Click or Ctrl + Click key combination for the desired tables.

2. Click >.

Alternatively, you can double-click on an item to move it. The selected tables appear in Tables in Data Model.

3. Click >> to add all available tables to your data model.

All available tables appear in Tables in Data Model.

4. To remove a table from the data model:

a. Select the table in Tables in Data Model.

b. Click <. Or you can double-click on an item to remove it. Alternatively, you may also click << to remove all tables from your data model.

iPlanet Application Builder displays all of your selected data source tables in the data model window.

Viewing or Editing Data Source Table Properties

You can view the properties of your data source tables to determine how to specify the interrelationships between your data model tables. In addition, if you import a data model or if you're moving from an application testing phase to a production phase, you can modify the database and user name while viewing the table properties. In the case of an imported data model, the `.pdm` file doesn't specify the database and user name information, so you must supply it; in the latter case, your production database is probably different from the one you use for testing and debugging your application, so this feature enables you to make the required changes without having to modify your original data model.

To view or change the data source table properties, perform the following steps:

1. Select the table handle (the top area that includes the table name).
2. If iPlanet Application Builder displays the properties in the Properties window, go to step 3; otherwise, right-click Properties; or from the Edit menu, choose View, then choose Properties.
3. Type a data source in Data Source.
4. Type a database name in Database, if applicable.
5. Type a user name in User.

Viewing a Column's Properties or Changing the Column Alias

You can view the column properties of your database tables in case you don't know whether a particular field is a number or string type. If you are creating a condition, you have to know whether to add single quotes, since string, date, time, and datetime data types must be single quoted. You may also want to know what the alias is for a specific column.

You use a **column alias** for a column name to rename the column title when you use a SELECT statement in a query. iPlanet Application Builder uses the column alias for referencing calculated columns and joins in hierarchical queries.

To use iPlanet Application Builder to view the database column properties or to change a database table's column alias, perform the following steps:

1. Select the Columns field.
2. In the properties window, type or edit the column alias in Column Alias.

Specifying Relationships Between Data Source Tables

You use iPlanet Application Builder to specify the relationships between the tables in your data model.

A **relationship** is a named connection between data source tables. For example, you might create a relationship between the `customer` and `order` tables by using a field common to both tables such as `customerID`. When you run a SELECT query, the application returns a result set in which key fields, such as customer and order records with identical `customerID` values, are combined into a single record.

Creating Relationships

To create a relationship between tables, perform the following steps:

1. Display the Properties window.
2. Select the desired table field and drag it onto a field in another table.

   iPlanet Application Builder automatically creates a one-to-many relationship between the two fields with the dragged column as the master column and the drop column as the detail column. This relationship is represented by a line linking the two fields. Perform step 2 if you want to change this default relationship type; otherwise, go to step 3.
3. Set the desired relationship type.

The following table describes the types of relationships you can use between your data model tables:

Table 4-1 Relationship Types

Relationship Type	**Description**
1-N	One-to-many
N-1	Many-to-one
1-1	One-to-one

To change the relationship type, perform the following steps:

1. Click on the desired relationship line between the two table fields. iPlanet Application Builder displays the SQL join information within the Properties window.

2. Click the Relationship drop-down box within the Properties window and choose the desired relationship type.

   iPlanet Application Builder creates the specified relationship between the two table fields.

3. If your relationship requires more than one column to establish a unique value, repeat step 1.

   For example, say you have three tables: *Customer*, *Product*, and *Sales*. *Customer* has only `CustID` and other information such as name, address, phone number. *Product* has only `ProductID` and other information such as product name, price, and amount in inventory. *Sales* has `CustID`, `ProductID`, `SalesID`, and `AmountSold`.

   In this example it is clear that you must link `Customer.CustID = Sales.CustID and Product.ProductID = Sales.ProductID.`

   So, the *Sales* table has multiple relationships with the other tables.

   If the *Customer* table didn't have a `CustID` field, but instead had only `FirstName` and `LastName` fields, then you would need two links from the *Customer* table to the Sales table. `Customer.FirstName = Sales.FirstName` and `Customer.LastName = Sales.LastName.`

   In this case, the *Customer* table requires two relationships with the *Sales* table in order to establish a unique value.

   If a field needs to join another field in the same table or if you want several fields in a table to join with the same field in another table, you can create an alias. To create an alias, right-click on the table and select Insert Table Alias from the pop-up menu. For an example, see "Creating Aliases".

Modifying or Deleting Relationships

You can use the iPlanet Application Builder Properties window to modify or delete relationships.

To modify the relationship, use the iPlanet Application Builder Properties window and modify the relationship as desired.

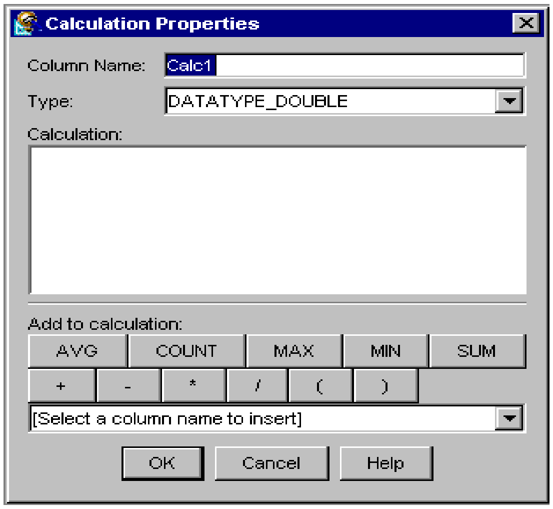
To remove a relationship, select the relationship you want to remove, and then press the Delete key, choose Delete from the Edit menu, or press the Backspace key.

Adding Calculated Fields

To reuse calculations across multiple queries in your application:

1. Select the table you want add a calculated field.
2. Right-click to display the table menu.
3. Select Insert Calculated Field.

   iPlanet Application Builder displays the calculation properties dialog box:



Column Name: Calc1
Type: DATATYPE_DOUBLE

4. Type the name of the calculated field in Column Name.
5. Select the type of the calculated field.

   The following data types are available:

Table 4-2 Data Types

Data Type	Description
Float	A decimal number, or number too large to fit in a long integer.
Integer	An integer.
Numeric	A roman numeral.
VarChar	A series of readable characters.
TimeStamp	Month, day, year, hours, minutes, seconds, and nanoseconds.

6. Type the specific calculation formula you want to assign to this field in the Calculation edit box. You can use calculation formulas similar to the following examples:

- count(*)
- sum(Sales.cost_of_goods_sold)
- avg(Sales.price * Sales.units_sold) / 1000

iPlanet Application Builder also enables you to make the following additional changes to a calculation:

- add a built-in function to a calculation
- add an operator or delimiter to a calculation
- include additional table columns within a calculation

The associated procedures for editing calculations are described in the following sections.

Adding a Built-in Function to a Calculation

To add a built-in function to your calculation, you can type it in the Calculation box or click on the specific function displayed in the Add to Calculation portion of the dialog box.

The built-in functions are described in the following table:

Table 4-3 Built-in Functions

Function Name	Description
AVG	Computes the average value of specified items.
COUNT	Computes the total number of specified items.
MAX	Computes the maximum value of the specified items.
MIN	Computes the minimum value of the specified items.
SUM	Computes the sum of all specified values.

Adding an Operator or Delimiter to a Calculation

To add an operator or delimiter to your calculation, you can type it in the Calculation edit box or click on the specific item displayed in the Add to Calculation portion of the dialog box.

The built-in operators are described in the following table:

Table 4-4 Built-in Operators

Name	Symbol
Addition	+
Subtraction	–
Multiplication	*
Division	/
Opening Parenthetic Delimiter	(
Closing Parenthetic Delimiter	)

Including Additional Table Columns Within a Calculation

To include additional table columns within the calculation, click on the column combo box displayed just below the operators within the Add to Calculation portion of the dialog box. iPlanet Application Builder adds each column name that you select to the calculation you create.

Importing a Third-Party Data Model

You can also import a Powersoft Physical Data Model (a file of type `.pdm`) from a third-party application. iPlanet Application Builder imports and displays the `.pdm` file as a `.kdm` file in the data model window.

To import an existing data model from a third-party application, perform the following steps:

1. From the File menu, choose Open File.
2. Find the data model you want to open.

   iPlanet Application Builder displays the data model that you've selected in the data model window.

Creating Queries

iPlanet Application Builder enables you to design several types of standard queries for accessing and/or modifying the data within your data source. You can select, insert, update, or delete information accessed from your data source.

You create your queries by first constructing or importing a data model that includes tables and the relationships between them from a specific portion of your data source. For a complete description of how to use iPlanet Application Builder to create a data model, see “Creating a Data Model.”

iPlanet Application Server applications can include four standard query types, as described in the following table:

Table 4-5 Query Types

Query Type	Description
`SELECT` query	Retrieves selected information from your data source, as specified by your data model.
`INSERT` query	Adds specific information to your data source.
`UPDATE` query	Modifies specific information within your data source.
`DELETE` query	Deletes specific information from your data source.

To create a new query, perform the following steps:

1. From the File menu, choose New, then choose Query.
2. Enter the desired information and click Next.
3. Choose a data model to use as the basis for this query and click Next. For more information on data models, see “Creating a Data Model.”

   You are now ready to create a specific type of data source query.
4. The New Query wizard displays the Create a New Query – Initial Query Type dialog box. Refer to the following sections that detail how to create the type of query you require.

This section includes the following topics:

- Creating a SELECT Query
- Creating an INSERT Query

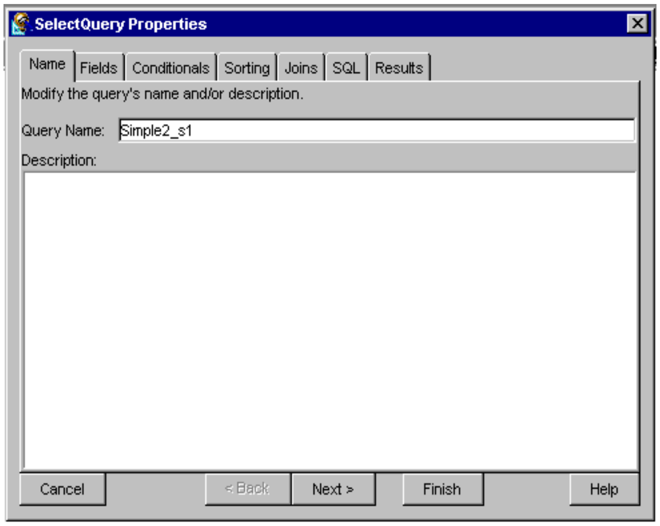
- Creating an UPDATE Query
- Creating a DELETE Query
- Testing Queries

Creating a SELECT Query

You use a SELECT query to retrieve information (columns) from specific tables within your data source.

To create a SELECT query, perform the following steps:

1. Click Select in the “Create a New Query – Initial Query Type” dialog box and click Continue (or click the Insert Select Query toolbar button; or choose Insert, and then choose Select Query). iPlanet Application Builder displays the following dialog box:



Modify the query's name and/or description.

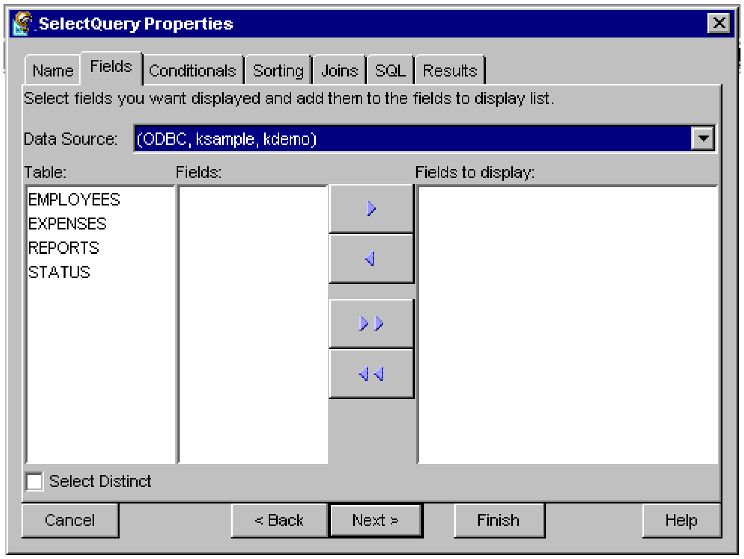
Query Name: Simple2_s1

Description:

As you use iPlanet Application Builder to create a SELECT query, you use the tabs displayed within the upper portion of the Select Query Properties dialog box. While you are filling out information in the tabs of this dialog box, iPlanet Application Builder automatically generates the corresponding query file.

Note that you can view and/or edit the generated query at any time by clicking the SQL tab located along the top of the query properties dialog box, or by using the iPlanet Application Builder properties window. For information that describes how to edit your queries, see “Editing Generated Code: Queries”.

2. Type a name for your query in Query Name, then click Next. iPlanet Application Builder displays the Select Query Properties Fields information:



Select fields you want displayed and add them to the fields to display list.

Data Source: (ODBC, ksample, kdemo)

Table: EMPLOYEES, EXPENSES, REPORTS, STATUS

Fields:

Fields to display:

Select Distinct

Follow these steps for each table containing fields you want to display:

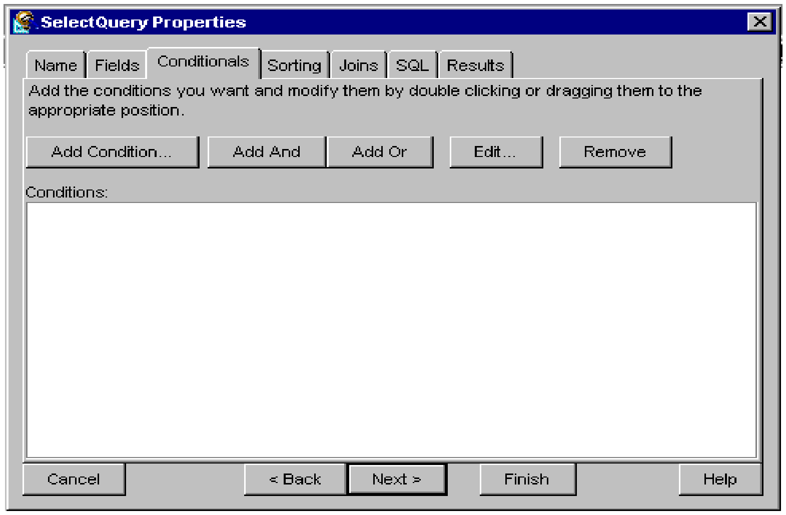
1. Select a table from the Table box. The available fields for that table appear in the Fields box.

2. Move fields into or out of the Fields to Display box by clicking the arrow icons. The double-arrow icons move all fields from one box to the other. Fields that appear in the Fields to Display box are selected for display by the query.

3. For query fields, check the Select Distinct checkbox if you want to perform a restricted search for all fields. A `SELECT DISTINCT` query retrieves only the unique instances of the requested target search items (for example, if you were searching for customers from different states, and your data source contained two customers named “Jerone Garcia,” both from California, the select distinct query would only return the first instance). For more information, consult the SQL reference for your data source.

4. You can specify conditional behavior for the query by adding conditions.

    a. Click the Conditionals tab on the Select Query Properties dialog box. iPlanet Application Builder displays the information displayed in the following illustration:



Add the conditions you want and modify them by double clicking or dragging them to the appropriate position.

Conditions:

b. Add a condition by clicking Add Condition.

c. Build the condition by selecting the table and field to use, selecting a conditional operator, and entering a value to limit the field's scope.

   Note that if a specific field is preceded with a colon (":"), iPlanet Application Builder displays the default field values for the following operators:

   =

   <>

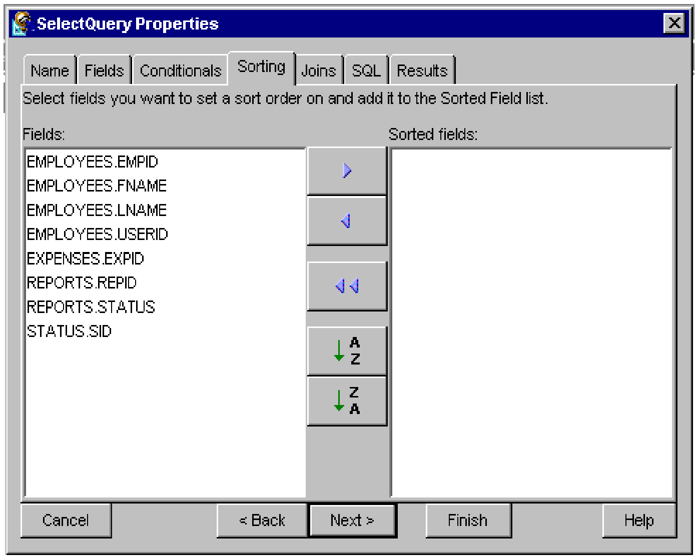
   <=

   >=

   For information about these operators, see "Simple Conditional Operators."

5. You can sort the results of your query by specifying which fields to sort and moving them into the Sorted Fields box.

6. Click the Sorting tab on the Select Query Properties dialog box. iPlanet Application Builder displays the associated data source fields:



Select fields you want to set a sort order on and add it to the Sorted Field list.

EMPLOYEES.EMPID
EMPLOYEES.FNAME
EMPLOYEES.LNAME
EMPLOYEES.USERID
EXPENSES.EXPID
REPORTS.REPID
REPORTS.STATUS
STATUS.SID

7. Change the sort order on a given field by clicking the Ascending (A to Z) or Descending (Z to A) button. For additional information that describes how to sort your query results, see "Sorting Query Results."

8. Fields must be joined in order to show the relationship between tables in the selection. For each set of fields to be joined, click the cell between them in the Join column and select the appropriate operator. For additional information regarding the possible join conditionals, see "Adding Conditionals to a Query."

9. Click the SQL tab to review the generated SQL (or you can edit the query directly). For information that describes how to edit your queries, see "Editing Generated Code: Queries."

10. To test the generated SQL, see "Testing Queries."

11. Click Finish. iPlanet Application Builder adds the query to your project and displays it on the main window. The properties for your query are shown in the Properties window, if it is visible.

Creating an INSERT Query

You use an `INSERT` query to add information to your data source.

To create an `INSERT` query, perform the following steps:

1. Click Insert in the "Create a new query" dialog box (or click the Insert Query toolbar button; or choose Insert, and then choose Insert Query).

   As you use iPlanet Application Builder to create an `INSERT` query, you use the tabs displayed within the upper portion of the Insert Query Properties dialog box. While you are filling out information in the tabs of this dialog box, iPlanet Application Builder automatically generates the corresponding query file.

   Note that you can view and/or edit the generated query at any time by clicking the SQL tab located along the top of the query properties dialog box, or by using the iPlanet Application Builder properties window. For information that describes how to edit your queries, see "Editing Generated Code: Queries."

2. Choose a name for your query, then click Next.

3. Follow these steps for each table containing fields you want to display:

   ❍ Select the data source.

   ❍ Select a table from the Table box. The available fields for that table appear in the Fields box.

   ❍ Type the specific value for each field in Value. Note that iPlanet Application Builder displays the default values for selected fields in quotes for string, date, time, and datetime field types.

   ❍ Click Next.

4. Click the SQL tab to review the generated SQL.

   For information that describes how to edit your queries, see "Editing Generated Code: Queries."

5. To test the generated SQL, see "Testing Queries."

6. Click Finish.

iPlanet Application Builder adds the query to your project and displays it on the main window. The properties for your query are shown in the properties window, if it is visible.

Creating an UPDATE Query

You use an `UPDATE` query to update information for specific data source tables.

To create an `UPDATE` query, perform the following steps:

1. Click Update in the "Create a new query" dialog box (or click the Insert Update Query toolbar button; or choose Insert, and then choose Update Query).

   As you use iPlanet Application Builder to create an UPDATE query, you use the tabs displayed within the upper portion of the Update Query Properties dialog box. While you are filling out information in the tabs of this dialog box, iPlanet Application Builder automatically generates the corresponding query file or method calls.

   Note that you can view and/or edit the generated query at any time by clicking the SQL tab located along the top of the query properties dialog box, or by using the iPlanet Application Builder properties window. For information that describes how to edit your queries, see "Editing Generated Code: Queries."

2. Choose a name for your query, then click Next.
3. Follow these steps for each table containing fields you want to display:
   - Select the data source.
   - Select a table from the Table box. The available fields for that table appear in the Fields box.
   - Type the specific value for each field in Value. Click Next.
4. You can specify conditional behavior for the query by adding conditions.
   - Add a condition by clicking Add Condition.
   - Build the condition by selecting the table and field, selecting a conditional operator, and entering a value to delimit the field's scope. For additional information regarding the possible join conditionals, see "Adding Conditionals to a Query."
5. Click the SQL tab to review the generated SQL. For information that describes how to edit your queries, see "Editing Generated Code: Queries."
6. To test the generated SQL, see "Testing Queries."
7. Click Finish. iPlanet Application Builder adds the query to your project and displays it on the main window. The properties for your query are shown in the properties window, if it is visible.

Creating a DELETE Query

You use a DELETE query to delete information from specific tables within your data source.

To create a `DELETE` query, perform the following steps:

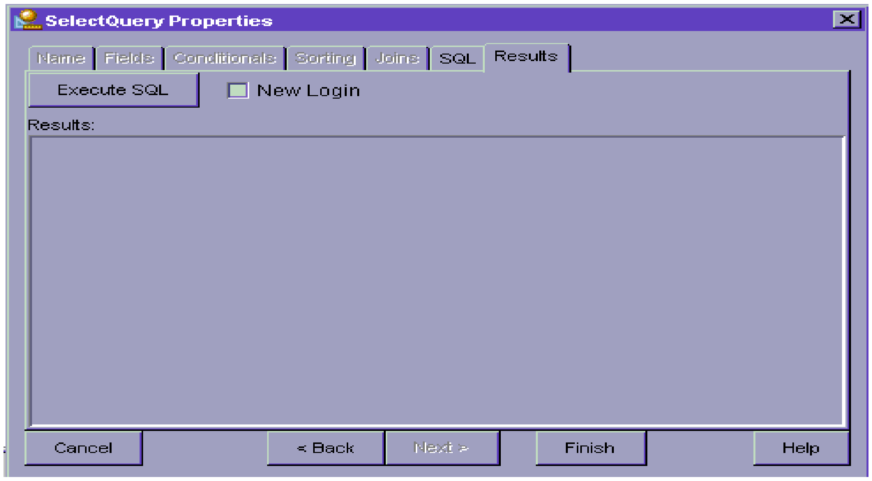
1. Click Delete in the “Create a new query” dialog box (or click the Insert Delete Query toolbar button; or choose Insert, and then choose Delete Query).

   As you use iPlanet Application Builder to create a `DELETE` query, you use the tabs displayed within the upper portion of the Delete Query Properties dialog box. While you are filling out information in the tabs of this dialog box, iPlanet Application Builder automatically generates the corresponding query file or method calls.

2. Choose a name for your query, then click Next.
3. Follow these steps for each table containing fields you want to delete:
   a. Select the data source.
   b. Select a table from the Table box and click Next.
4. You can specify conditional behavior for the query by adding conditions. For additional information regarding the possible join conditionals, see “Adding Conditionals to a Query”.
   a. Add a condition by clicking Add Condition.
   b. Build the condition by selecting the table and field, selecting a conditional operator, and entering a value to delimit the field’s scope.
5. Click the SQL tab to review the generated SQL. For information that describes how to edit your queries, see “Editing Generated Code: Queries”.
6. To test the generated SQL, see “Testing Queries”.
7. Click Finish. iPlanet Application Builder adds the query to your project and displays it on the main window. The properties for your query are shown in the properties window, if it is visible.

Testing Queries

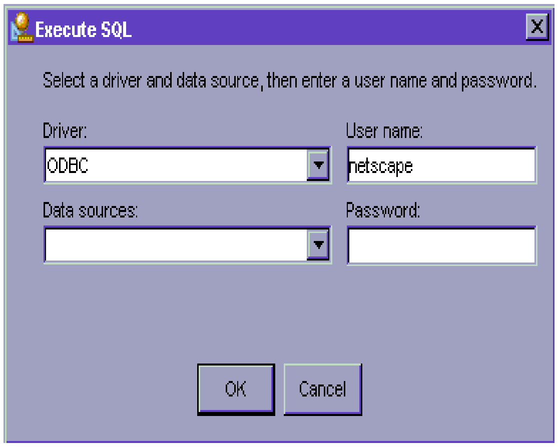
Once you’ve created your queries, you can use iPlanet Application Builder to test them and view the results using the Results tab.



1. Select New Login to use a different login name than was used in the previous query.

2. Select Execute SQL to identify the driver and data source.

   The following dialog box appears:


Select a driver and data source, then enter a user name and password.

Driver: ODBC

User name: netscape

Data sources:

Password:

3. Select a driver and data source. Also, identify the username and password.

   The results of the query are displayed in the results field.

For more information about how to test queries, see in “Compiling, Testing, and Debugging Applications.”

Editing Generated Code: Queries

You can edit the SQL code that you create (or that iPlanet Application Builder automatically creates) via the SQL tab on the query properties dialog box.

Editing Queries via the SQL Tab

You can edit the SQL code that makes up your query by selecting the SQL tab in the Query Properties dialog box. This tab displays the query file or SQL that iPlanet Application Builder is generating in response to your selections in the other query creation dialog tabs.

To edit a specific query using the Query Properties SQL tab, perform the following steps:

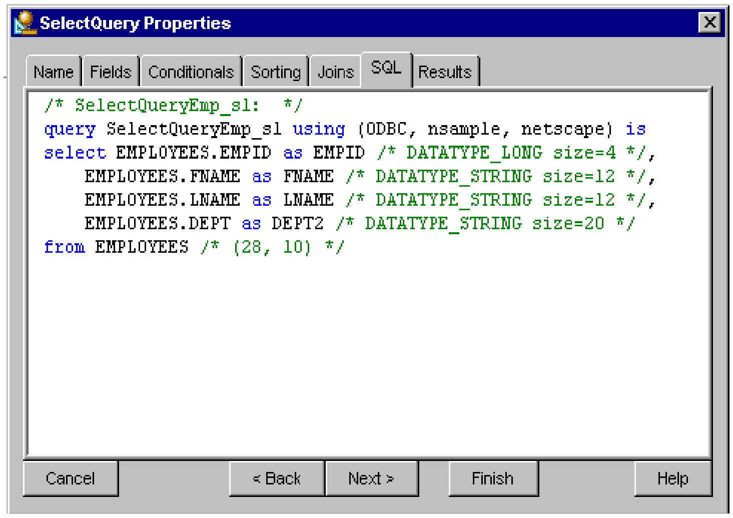
1. From the File menu, choose Open to open the desired query file or double-click the query file in the project map window or within the Project window.
2. Right-click the specific query you want to edit within the Outline tab and select the Goto option. iPlanet Application Builder displays the Source tab.
3. Edit the generated SQL.

   Note that table and field names are case sensitive and must match the case of the corresponding names in the data model.

To preview your query SQL code, perform the following steps:

1. From the File menu, choose Open to open the desired query file or double-click the query file in the project map window or within the Project window.
2. Click the Query Properties SQL tab.

   iPlanet Application Builder displays the associated SQL code for the specified query within the Query Properties SQL dialog box, as shown in the following illustration:



```
/* SelectQueryEmp_sl:  */
query SelectQueryEmp_sl using (ODBC, nsample, netscape) is
select EMPLOYEES.EMPID as EMPID /* DATATYPE_LONG size=4 */,
    EMPLOYEES.FNAME as FNAME /* DATATYPE_STRING size=12 */,
    EMPLOYEES.LNAME as LNAME /* DATATYPE_STRING size=12 */,
    EMPLOYEES.DEPT as DEPT2 /* DATATYPE_STRING size=20 */
from EMPLOYEES /* (28, 10) */
```

NOTE If you add incorrect or unsupported SQL code, iPlanet Application Builder disables the other available tabs until you correct the error.

Editing Query Descriptions

You can edit query descriptions that appear in the Properties window. To edit a specific query description, perform the following steps:

1. From the File menu, choose Open to open the desired query file or double-click the query file in the project map window or within the Project window.

2. Click the specific query you want to edit within the left portion of the query window.

   iPlanet Application Builder displays the following information within the Properties window:

   - query name
   - description
   - type

3. Make the desired changes within the properties window.

   The displayed information will be read-only if custom SQL is present.

Adding Conditionals to a Query

You can specify conditions that are used to match rows in the data source to help focus your query to obtain more precise search results. For example, rather than retrieving information about all customers, you can use a condition to retrieve only information about customers in a certain city.

The Conditionals tab of the Query Properties dialog box allows you to set up the `WHERE` clause of a SQL query.

The `WHERE` clause of a SQL query can take multiple forms:

- Simple condition.

  For example: `WHERE lastName='Garcia'`

- Boolean clause. This clause can be a set of simple conditions linked together by a logical Boolean operator (AND, OR):

  For example: `WHERE lastName='Garcia' OR city='half moon bay' OR acctNum=1234567890`

  A boolean clause can consist of any combination of simple conditions and other boolean clauses linked together by a logical Boolean operator. For example: `WHERE lastName='Garcia' OR city='half moon bay' OR (firstName='Jerone' AND homePhone='999-999-9876')`.

  Note that parentheses are used in a condition statement to force iPlanet Application Builder to interpret specific conditions first. In this example, the `AND` condition must be interpreted prior to the preceding `OR` conditions.

Choosing Between Simple and Boolean Conditional Operators

The Conditionals tab displays both simple conditions and boolean clauses in a tree format. The parent nodes within this tree represent boolean operators which form boolean clauses by linking together the child nodes representing simple conditions and the child subtrees representing boolean clauses.

Simple Conditional Operators

iPlanet Application Builder includes a complete set of simple conditional operators, as described in the following table:

Table 4-6 Simple Conditional Operators

Name	Symbol	Description
Equals	=	Specified field and value is equivalent
Not equals	<>	Specified field and value is not equivalent
Less than	<	Specified field is less than value
Less than or equal to	<=	Specified field is less than or equal to value
Greater than	>	Specified field is greater than value
Greater than or equal to	>=	Specified field is greater than or equal to value
Is between	IS BETWEEN	Specified field value is between value
In	IN	Specified field is part of the value set
Not in	NOT IN	Specified field is not part of value set
Is null	IS NULL	Specified field is null
Is not null	IS NOT NULL	Specified field is not null
Is like	IS LIKE	Specified field is similar to value

iPlanet Application Builder displays these simple conditional operators within the Conditions Property dialog box, as seen when you add a condition to a query.

Boolean Conditional Operators

iPlanet Application Builder provides two boolean conditional operators within the Conditions Property dialog box: `AND` and `OR`.

The AND boolean operator returns TRUE if and only if both operands are TRUE.

The OR boolean operator returns TRUE if one or both operands are TRUE.

MS SQL SmallDateTime and DateTime Columns

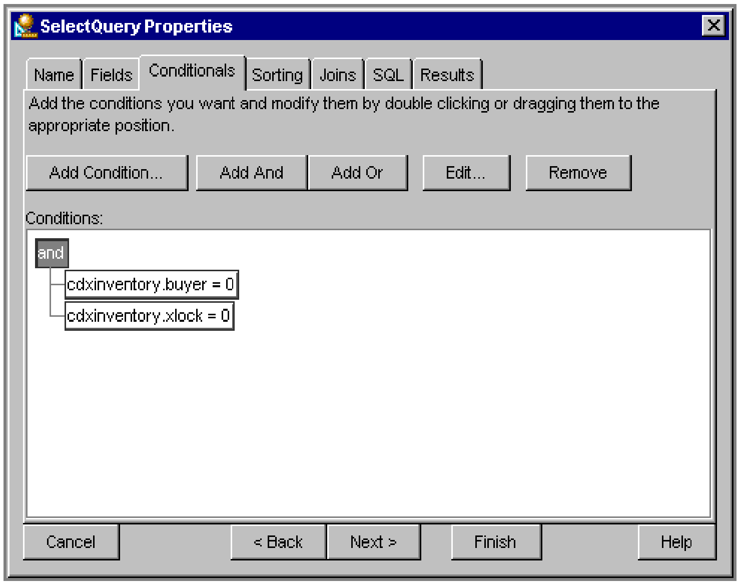
If you retrieve the data type for smalldatetime or datetime columns in MS SQL, iPlanet Application Builder maps the actual data type (type 11) to the ODBC Timestamp data type (type 93).

Adding a Conditional Operator to a Query

To add a conditional operator to a query, perform the following steps:

1. From the File menu, choose Open to open the desired query file or double-click the query file in the project map window or within the Project window.
2. Click the Query Properties Conditionals tab.

   iPlanet Application Builder displays the Query Properties Conditionals dialog:



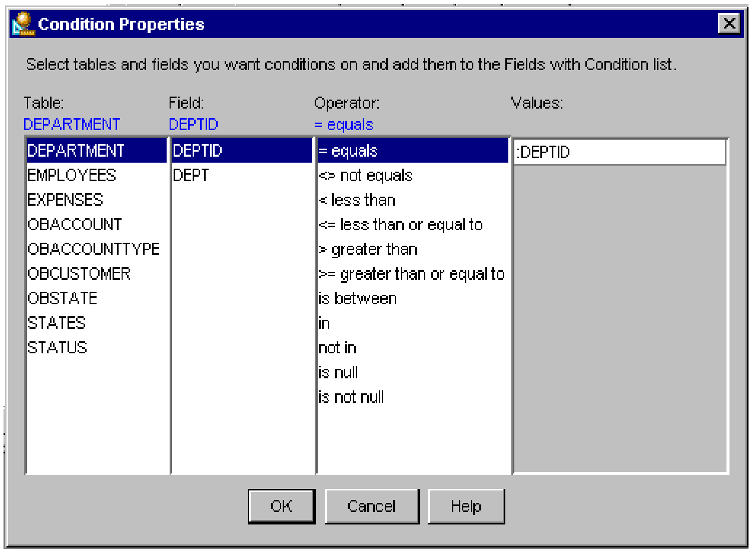
Add the conditions you want and modify them by double clicking or dragging them to the appropriate position.

Conditions:

and
cdxinventory.buyer = 0
cdxinventory.xlock = 0

3. Click Add And or Add Or to add an AND or OR operator.
4. Click Add Condition to add a condition to the AND or OR operator.

   iPlanet Application Builder displays the Conditionals Properties dialog box:


Condition Properties

Select tables and fields you want conditions on and add them to the Fields with Condition list.

Table:	Field:	Operator:	Values:
DEPARTMENT	DEPTID	= equals	
DEPARTMENT	DEPTID	= equals	:DEPTID
EMPLOYEES	DEPT	<> not equals	
EXPENSES		< less than	
OBACCOUNT		<= less than or equal to	
OBACCOUNTTYPE		> greater than	
OBCUSTOMER		>= greater than or equal to	
OBSTATE		is between	
STATES		in	
STATUS		not in	
		is null	
		is not null	

5. Select the specific table, fields, and operators.

For a list of the available operators and their definitions, see “Simple Conditional Operators.”

6. Click in the text field in Values.

The Conditionals Properties dialog box highlights the Values text field.

7. Type the required values in the text boxes. Note that iPlanet Application Builder displays the default query parameter.

For example, if you selected “equals to”, type a value which the data source field must match, such as 100 or ‘Hanan’.

Note that single quotes are required around string and date values.

Using Query Parameters

The Query Properties Conditionals tab allows you to instruct a query to accept input from the calling servlet.

This function is achieved by entering `:<variable>` (such as a `valIn`, session variable, or constant) as you work with the Query Properties Conditional tab.

When you create a Data Access servlet and add a query, you are prompted for query parameters. You can pass in a `valIn` variable or a session variable, or you can enter a constant for the query parameter. For a complete description of servlets, see "Editing Servlets" in "Creating Presentation Logic."

Adding Joins to a Query

You can specify joins that define how queries relate to each other. A **join** sets the relationship between two tables or two queries.

iPlanet Application Builder provides the following types of join operators:

Table 4-7 Join Operators

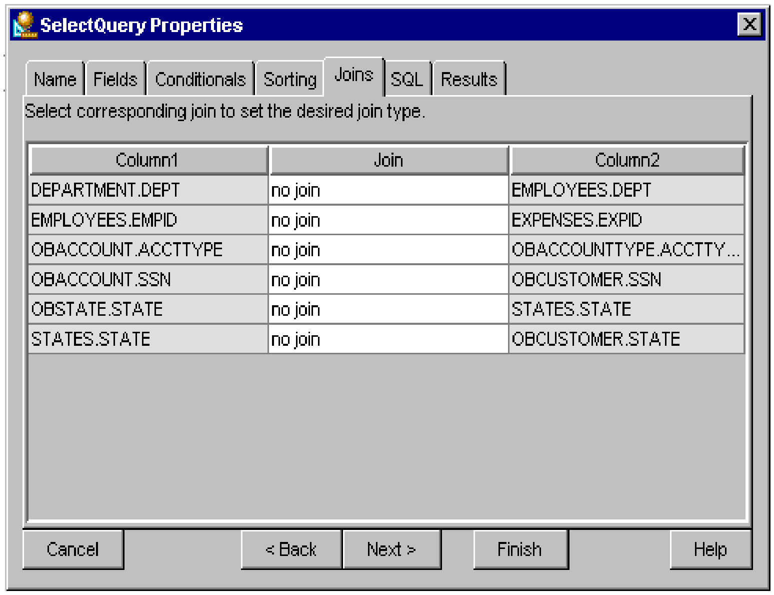
Name	Symbol	Description
Simple join	=	Returns rows from two tables based on an equality condition.
Left join	(+) =	Returns rows from two tables based on the equality condition plus those unique rows from the left table.
Right join	= (+)	Returns rows from two tables based on the equality condition plus those unique rows from the right table.
No join	No join	Specifies that no joins are used in the table.

Joins can be made only if relationships have been defined in the associated data model. For a complete description of how to define relationships within a data model, see "Specifying Relationships Between Data Source Tables".

To add a join to a query, perform the following steps:

1. From the File menu, choose Open to open the desired query file, or double-click the query file in the project map window or within the Project window. The Select Query Properties window appears.
2. Click the Joins tab on the Query Properties dialog box.

   iPlanet Application Builder displays the Query Properties Joins dialog box:



Select corresponding join to set the desired join type.

Column1	Join	Column2
DEPARTMENT.DEPT	no join	EMPLOYEES.DEPT
EMPLOYEES.EMPID	no join	EXPENSES.EXPID
OBACCOUNT.ACCTTYPE	no join	OBACCOUNTTYPE.ACCTTY...
OBACCOUNT.SSN	no join	OBCUSTOMER.SSN
OBSTATE.STATE	no join	STATES.STATE
STATES.STATE	no join	OBCUSTOMER.STATE

3. For each column, click the Join field to select the desired join operator.
4. Click Next to continue editing the selected query, or click Finish if you have finished editing the query.

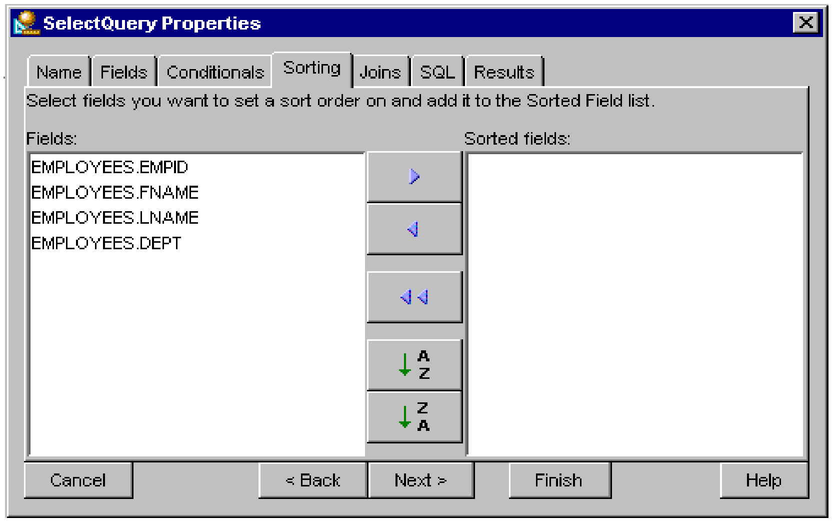
Sorting Query Results

You can change the order of the rows retrieved by a query. Normally, when you work with retrieved data, you want it to be organized in a particular sequence. This sequence is determined by a **sort key**, which is a group of one or more column names.

If no field names appear in this tab, go back and fill out the appropriate Tables and Fields tabs. Those tabs must be completed before you can use the Sorting tab.

To sort query results, perform the following steps:

1. From the File menu, choose Open to open the desired query file, or double-click the query file in the project map window or within the Project window.
2. Click the Sorting tab on the Query Properties dialog box.



Select fields you want to set a sort order on and add it to the Sorted Field list.

Fields:

EMPLOYEES.EMPID
EMPLOYEES.FNAME
EMPLOYEES.LNAME
EMPLOYEES.DEPT

Sorted fields:

3. In Fields, select one or more field names that you want to use as a sort key.

   To select multiple fields, click one field name after another.

4. Click >.

   Application Builder adds the selected field to the Sorted Fields list. The default sort order is A–Z (ascending order). To change the field sort order to Z-A (descending order), click Sort (Z–A), located between the Fields and Sorted Fields lists. You can also select a field, then click Sort (A–Z) or click Sort (Z–A) to add that field to the Sorted Fields list in the specified sort order.

5. To change the sort order on a particular column:

   a. Select the field name in Sorted Fields.

   b. Click Sort (A-Z) or Sort (Z-A), for ascending or descending order.

6. To remove a field from the sort key:

   a. Select the field name in Sorted Fields.

   b. Click <.

      You can also remove all the fields by clicking <<.

NOTE While executing a query string, parameters must be specified with single quotation marks 'xxx'.

Chapter 5

Creating Presentation Logic

This chapter describes the concepts and tasks associated with creating the presentation logic for web applications and describes how to use iPlanet Application Builder's Java editor and Properties window to customize servlets.

The following topics are described in this chapter:

About Java Servlets

A Java **servlet** contains the application code that executes on the server in response to an event. A servlet receives events from the web application page, performs the appropriate actions, and then subsequently outputs data dynamically to the application's presentation layout model by invoking a JavaServer Page (JSP).

Servlets have the following characteristics:

- They are an alternative to a CGI script.

- They are invoked by a browser/client request.
- They communicate using request and a response objects.
- They output content via HTML streaming.
- They simplify HTTP programming.
- They control the application by specifying the flow of control, for example, which enterprise components to activate, what to cache, and which HTML pages or JSP files to invoke.
- They are portable.

A servlet does not necessarily have to be designed for one specific application; you could create a library of servlets to be used across multiple applications. There are advantages to writing application-specific servlets, however, especially in terms of application configuration issues.

For a more in-depth description of how to design and create servlets programmatically, see the *Programmer's Guide.*

How Servlets Control the Presentation Logic

The design of the application's presentation, data access, and business logic help to determine the design of each servlet. In addition, the number and type of the servlets required to control the presentation logic can be designed from the point of view of the possible interactions a user may input while using the application.

A user's **interaction** with an application takes place via JavaServer Pages (JSPs). Information entered on a page form is sent to the servlet in a request object and is processed bythe servlet's `doGet()` and `doPost()`methods. The servlet then sets up a response by performing any necessary business logic functions, such as accessing databases or directories or performing complex calculations on provided data. Business logic is usually performed by external objects, such as Enterprise JavaBeans (EJBs) and not by the servlet itself. For more information about EJBs, see "Creating Business Logic."

The servlet-generated response is an HTML page or a JSP that contains information associated with the request, which sets up the next interaction. This page can be immediately returned and displayed by the client object, or generated by invoking a JavaServer Page (JSP) which assembles the response and passes it back to the client browser.

Servlets Work with Various Types of Components

To control the application's presentation logic, servlets need to communicate with various types of additional project components. These servlet-related components include the following types of project files or APIs:

- JDBC RowSet Interface
- Session management mechanisms
- Enterprise JavaBeans
- JavaServer Pages

JDBC RowSet Interface

A JDBC `RowSet` object is a Java object that encapsulates a set of rows that have been retrieved from a database or other tabular data source, such as a spreadsheet. The `RowSet` interface provides JavaBean properties that allow a `RowSet` instance to be configured to connect to a data source and retrieve a set of rows.

For more information about the JDBC RowSet interface, see "Creating and Editing JDBC RowSet Objects" .

Session Management Mechanisms

A **session** represents a series of user-application interactions that are tracked by the server application. Tracking sessions enables an application to maintain a separate context for each user, and to provide security for such interactions (for example, a validated user login followed by a series of directed activities for that particular user).

iPlanet Application Builder enables you to track session information via the following mechanisms:

- iPlanet Application Server Standard (distributed)
- Servlet Standard (nondistributed)

For more information about session management and designing servlets that process session data via a session-tracking mechanism, see "Working with Session Management" .

Enterprise JavaBeans

Servlets act as the central dispatcher for your application and handle presentation logic; **Enterprise JavaBeans (EJBs)** do the bulk of your application's actual data and business rules processing. However, note that EJBs provide no presentation (actual page flow or layout, or the logic behind such) or visible user-interface services. Each EJB encapsulates one or more application tasks or application objects. Typically, they take parameters and send back return values.

For more information about Enterprise JavaBeans, see "Creating Business Logic."

JavaServer Pages

A **JavaServer Page (JSP)** is an HTML file, with Sun-specified extensions, that is executed on a web server or application server instead of on a client, dynamically creating pure HTML, which is then sent to the client. iPlanet Application Builder 6.0 has a set of JSP extended tag libraries. The wizard generated JSPs make use of these libraries for taks like database access. These libraries allow the author to embed scripts and expressions in any scripting language (the default is Java); perform NCSA-standard server-side includes (incorporating other JSPs from the server) conditionally or unconditionally, call Java Servlets, and call Enterprise Java Beans (EJBs) to perform tasks on the server, and generate output to be incorporated into the dynamic HTML stream.

By default, the JSPs generated through the JSP wizard are automatically registered. However, JSPs generated by all other wizards are not registered because they always have a registered servled associated with them that forwards the request object to the JSP for presentation.

For more information about JavaServer Page, see "Creating Presentation Logic."

Servlets

A servlet acts much like a "controller" in the MVC (Model / View / Controller) user interface architecture. A servlet receives user-generated events and directs changes to both the underlying "data storage" model, such as Enterprise JavaBeans, the iPlanet Application Server session information, and so on, and to the view you want to display, which is either an HTML page or a JavaServer page.

These user-generated events most often come from an HTML form. The server packages the arguments into the request object, and calls the servlet's main entry point `doGet()` and `doPost()` methods.

If successfully handled, the servlet processes the output, most often by calling a JSP page. The default page is stored in the generated servlet as `_defaultTemplate`. The servlet then returns the output page to the user.

Creating and Editing Servlets

iPlanet Application Builder provides a servlet wizard which generates a breed of servlets with added value that enable point and click data binding of servlet data with various types of components in JavaServer Pages (JSPs).

Use the various code generation wizards provided with iPlanet Application Builder to generate servlet code automatically. The wizards provide a framework that gives you a head start on the most commonly used types of servlets.

Consider using a Java IDE for coding your presentation logic in Java rather than using the built-in Java code editor, which is best suited for editing minor changes in the application code.

This section describes the following servlet-creation topics:

- Creating Servlets via the Wizards
- Editing Servlets
- Handling Servlet Output
- Importing Servlets
- Registering, Compiling, and Testing Servlets

Creating Servlets via the Wizards

To create your iPlanet Application Server servlets using wizards, perform the following steps:

1. Run the servlet wizard.
2. Edit the iPlanet Application Builder-generated code.
3. Compile the servlets.
4. Test the servlets.
5. Debug the servlets.

For information about compiling, testing, or debugging servlets, see “Compiling, Testing, and Debugging Applications.”

You can create new servlets or add additional code to an existing servlet by using wizards, or by editing the files directly. iPlanet Application Builder automatically generates Java code based on your selections or insertions.

Editing Servlets

This section describes the specific aspects of a servlet that you can edit using the various iPlanet Application Builder tools.

The following table maps the specific servlet components you can edit to the associated iPlanet Application Builder servlet editor tool.

Table 5-1 A Sampling of Servlet Components Mappings

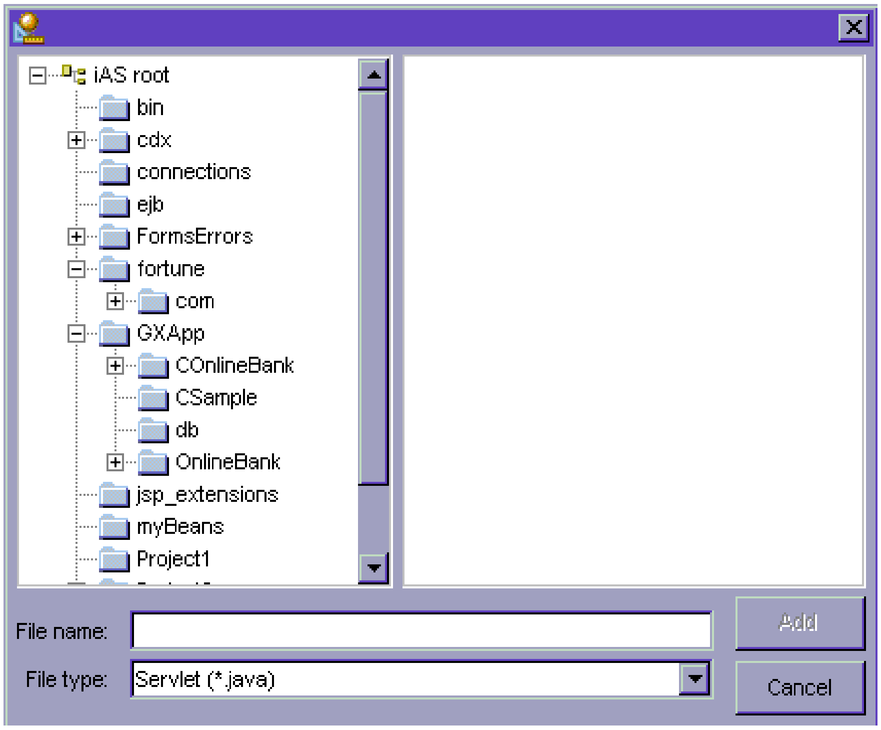
Servlet Component	iAB Editor Tool
Parameters	Meta Editor Dialog
Business Logic	Pure coding via the Java editor
Caching	Meta Editor Dialog

The Properties editor for a servlet specifies properties that are set when you create the servlet via a wizard. You can change the initial parameters, caching information, validation parameters, through the project meta editor. For more information about these properties, see the *Programmer’s Guide*.

Importing Servlets

To import a servlet, perform the following steps:

1. Copy the servlet file to a directory under the iAS root.
2. Select Import Servlet(s) from the Project menu.
3. Select the servlet file, either by navigating or by entering the file name, and click Add.



iPlanet Application Builder places the servlets in the correct location in the project tree, depending on whether the servlet is source or compiled code.

Registering, Compiling, and Testing Servlets

Once you have created your Java objects, you can use iPlanet Application Builder to register, compile and test them. When you test a Java object, your preferred HTML browser displays an HTML page from which you can submit a request to run the object.

For more information about how to register, compile, and test servlets, see "Compiling, Testing, and Debugging Applications."

Working with Session Management

A **session** is a continuous series of interactions between a user and a iPlanet Application Server application. The term session is widely used to refer to a web browser session, but in this manual,the term session refers more specifically to the user interactions that are tracked by a iPlanet Application Server application. The user's session with a web browser or other client software might start before the iPlanet Application Server application begins tracking the user, and could continue after the application stops tracking the user.

Sessions are useful when you want to store information about each user's interaction with the application, as illustrated in the following examples:

- Record a history of which pages the user has visited during the session.
- Increase security by requiring the user to log in to a secured session before running certain portions of the application.
- In an online shopping application, keep track of items in the user's shopping cart.

To track session information, a base session class uses a specific iPlanet Application Builder session interface (`HTTPSession`). Each time a particular servlet is executed during the session, the servlet can instantiate the base session, which accesses an `HTTPSession` interface provided by a iPlanet Application Server. This is a view of the actual session information. The session ID ensures that all these session instances in fact point to the same data.

This section describes the following topics:

- Storing Session Variables at Design Time
- Modifying Session Variables

Storing Session Variables at Design Time

The project file (`*.iab`) stores session information specific to your iPlanet Application Builder project. Servlets use sessions to store information about each user's interaction with the application. For example, a login servlet might create a session object to store the user's login name and password. This session data is then available to other servlets in the application.

Modifying Session Variables

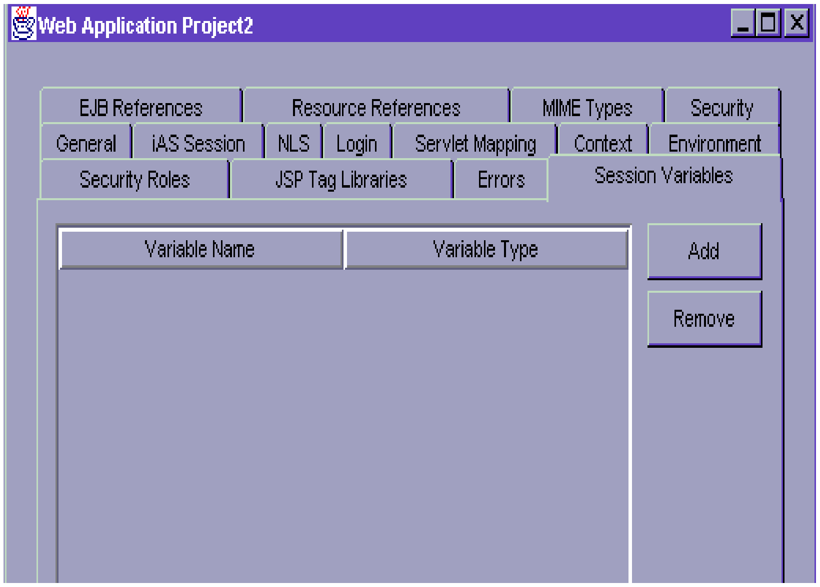
iPlanet Application Builder also helps you to create and manage session variables for your application.

Session variables are used by many servlets in an application to store and access data that is shared throughout the application. They behave much like global variables do for an application; if you think of servlets as separate procedures and functions for an application, the session variables act (per user) as globally accessible data blocks that are shared across all servlets.

For example, a banking application can store the balance of the current account in a session variable; as the user executes a series of servlet requests, the balance is updated on the server by manipulating a session variable. When you define a session variable with iPlanet Application Builder, the variable becomes automatically accessible for use by all iPlanet Application Builder wizards where input fields are required. This mechanism makes it easy to pass session variables into other servlet requests or to a query parameter.

To modify your session variables, perform the following steps:

1. Select the Project you want to modify.
2. Right-click to display a pop-up menu, and select Project Metadata ...
3. Select the Session Variable tab from the Metadata Editor window.

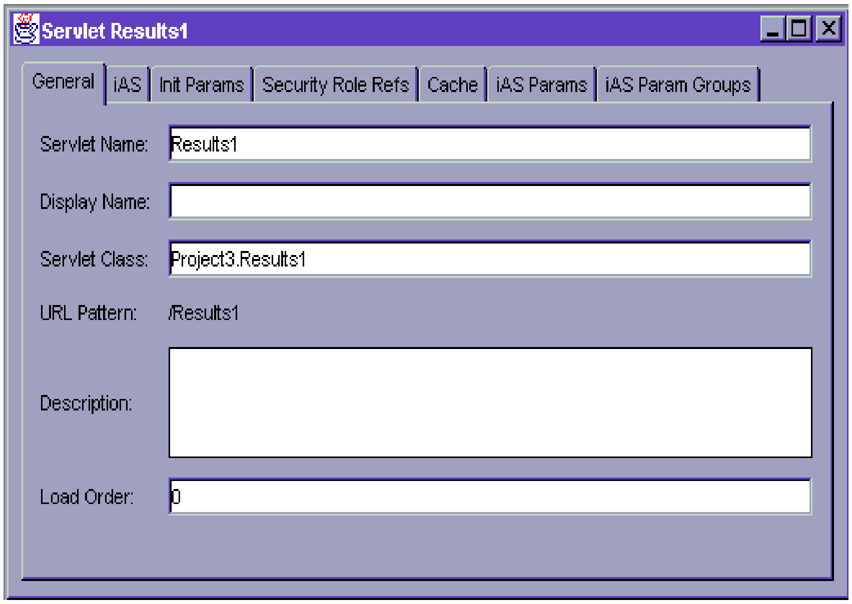


4. Add or remove variables from the list of design time session variables.

   The Variable Type column of this test is used to typecast the object returned from the session at runtime.

Validation

You can specify whether or not to validate a parameter and the kind of validation to enforce in the Servlet Meta Editor:



Servlet Name: Results1

Display Name:

Servlet Class: Project3.Results1

URL Pattern: /Results1

Description:

Load Order: 0

1. Select the Init Params tab.
2. You can add or delete parameters to be validated.

In addition to specifying validation in the Properties window, you can provide user validation. **User validation** is validation written by the user. Typically, user validation is for inter-field dependencies (such as, `orderDate < shippingDate < receivingDate`), or field validation which requires data source access (or some other form of complex access); for example, validating credit cards.

Servlet Parameter Validation

A typical request-response cycle goes through several phases, the first of which is the checking required to ensure that the user has supplied all the correct input, and that they are properly authenticated for the access they require.

Note that this validation is not trying to solve complex validation such as credit checks, although it is possible to do so.

Validation Requirements

The `ValidationRequired` flag for the servlet should be set to true.

Each parameter needs to be identified and have at least the following attributes:

- Session attribute or an input parameter
- Required parameter
- Class and Method to call in that class to determine the validity of the parameter
- Optional format (for numbers and dates)
- Definition of how errors should be handled including:
  - Java Server Page URL
  - Servlet
  - Log and continue setting (store failure but continue validation)

These fields are identified in the following Servlet Meta Editor Parameters tab:

The validation scheme itself needs a URL to call when errors have been accumulated. An error handler at the servlet level should be set.

Additionally, one servlet may perform different functions. You could create a servlet to perform a search/insert/delete against a particular data source, and another server performing different functions. While the parameters for both servlets may have the same requirements, the list of which parameters to validate would be different depending on the entry points. So, along with a list of parameters, a list of `ParamGroups` must be identified with each event mapping to a list of parameters to validate.

NOTE Exposing the class:method pairs for validation give you the ability to add reusable validation components.

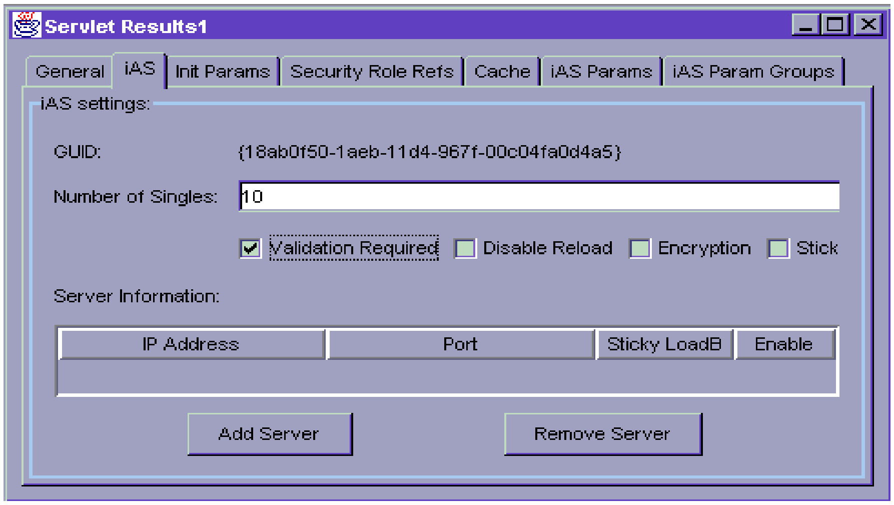
Parameter information is generated in the Login, Input, and Results wizard.

For more information about iAS specific DTD, see the *Programmer's Guide.*

Parameter and Parameter Groups

To enable servlet parameter validation:

- Check the Validation Required checkbox in the iAS tab of the Servlet Meta Editor:


Servlet Results1

GUID: {18ab0f50-1aeb-11d4-967f-00c04fa0d4a5}

Number of Singles: 10

[✓] Validation Required [ ] Disable Reload [ ] Encryption [ ] Stick

Server Information:

IP Address	Port	Sticky LoadB	Enable

You can specify an error handler at the servlet level as well as the parameter level. The parameter error handler can be specified as a JSP, servlet, or log file.

In the following example, if param-error-handler is specified as a log, the error handler at the servlet level is used (`Error.jsp`).

- The first servlet entry (`Errorjsp.jsp`) is a registered JSP for error handling.
- The second servlet entry (`ErrorServlet`) is a registered servlet for error handling.
- The third servlet is `TestParamGroup` whose parameters are grouped by `ParamGroups` and are to be validated.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE ias-web-app PUBLIC '-//Sun Microsystems, Inc.//DTD iAS Web
Application 1.0//EN'
'http://www.iplanet.com/j2ee/dtds/IASWebApp_1_0.dtd'>
<ias-web-app>
```

```
    <servlet>
       <servlet-name>Errorjsp</servlet-name>
       <guid>{86422ba5-15a8-11d4-b65a-005004b1b6f4}</guid>
       <servlet-info>
          <sticky>false</sticky>
          <encrypt>false</encrypt>
          <number-of-singles>10</number-of-singles>
          <disable-reload>false</disable-reload>
       </servlet-info>
    </servlet>

    <servlet>
       <servlet-name>ErrorServlet</servlet-name>
       <guid>{86422ba3-15a8-11d4-b65a-005004b1b6f4}</guid>
       <servlet-info>
          <sticky>false</sticky>
          <encrypt>false</encrypt>
          <number-of-singles>10</number-of-singles>
          <disable-reload>false</disable-reload>
       </servlet-info>
    </servlet>

    <servlet>
       <servlet-name>TestParamGroup</servlet-name>
       <guid>{86422ba4-15a8-11d4-b65a-005004b1b6f4}</guid>
       <validation-required>true</validation-required>
       <error-handler>Error.jsp</error-handler>
       <servlet-info>
          <sticky>false</sticky>
          <encrypt>false</encrypt>
          <number-of-singles>10</number-of-singles>
          <disable-reload>false</disable-reload>
       </servlet-info>

 <parameters>
    <param>
       <param-name>one</param-name>
       <input-fields>
          <input-required>true</input-required>
          <input-rule>com.iplanet.server.servlet.extension.
          validate.ValidationUtils.validateAlphabetic</input-rule>
          <in-session>false</in-session>
          <param-error-handler>ErrorServlet</param-error-handler>
       </input-fields>
    </param>
```

```
        <param>
            <param-name>two</param-name>
            <input-fields>
                <input-required>true</input-required>
                <input-rule>com.iplanet.server.servlet.extension.
                validate.ValidationUtils.validateInteger</input-rule>
                <in-session>false</in-session>
                <param-error-handler>ErrorServlet</param-error-handler>
            </input-fields>
        </param>

        <param>
            <param-name>three</param-name>
            <input-fields>
                <input-required>true</input-required>
                <input-rule>com.iplanet.server.servlet.extension.
                validate.ValidationUtils.validateInteger</input-rule>
                <in-session>false</in-session>
                <param-error-handler>log</param-error-handler>
            </input-fields>
        </param>

        <param>
            <param-name>four</param-name>
            <input-fields>
                <input-required>true</input-required>
                <input-rule>com.iplanet.server.servlet.extension.
                validate.ValidationUtils.validateInteger</input-rule>
                <in-session>false</in-session>
                <param-error-handler>log</param-error-handler>
            </input-fields>
        </param>

        <param>
            <param-name>five</param-name>
            <input-fields>
                <input-required>true</input-required>
                <input-rule>com.iplanet.server.servlet.extension.
                validate.ValidationUtils.validateInteger</input-rule>
                <in-session>false</in-session>
                <param-error-handler>Errorjsp.jsp</param-error-handler>
            </input-fields>
        </param>
    </parameters>
```

```
<param-group>
   <param-group-name>update</param-group-name>
   <param-input>one</param-input>
   <param-input>two</param-input>
   <param-input>three</param-input>
</pram-group>

<param-group>
   <param-input>four</param-input>
   <param-input>five</param-input>
</param-group>

</servlet>

</ias-web-app>
```

Accessing Data Sources via JDBC

In some situations, accessing a database directly from a servlet (via embedded JDBC) can be faster than accessing databases from EJBs. Fewer calls need to be made, especially if your application is spread across servers so that your EJBs are accessible only through the **Java Remote Method Interface (RMI)**.

Use direct database service through servlets as little as possible. If you do provide database access from servlets restrict access to those situations where access is very short duration, the transaction is read-only, and you can take advantage of the new JDBC 2.0 RowSet class. If you choose to access a database from a servlet, use the new JDBC 2.0 `RowSet` interface to interact with a database.

Creating and Editing JDBC RowSet Objects

A **row set** is an object that encapsulates a set of rows retrieved from a database or other tabular data store, such as a spreadsheet. To implement a row set, your code must import `javax.sql`, and implement the `RowSet` interface. `RowSet` extends the `java.sql.ResultSet` interface, permitting it to act as a JavaBeans component.

For more information about the `RowSet` interface, see the *Programmer's Guide.*

Setting Data Connections

You must specify a data connection when you run the database wizards. The wizard prompts you for the following information:

Table 5-2 Data Connection Wizard Prompts

Property	Description
Driver	The database driver with which to connect to the data source. Choose from one of the following drivers: • INFORMIX_CLI • DB2_CLI • SYBASE_CTLIB • ORACLE_OCI • ODBC
Data sources	The database or data source name.
User name	The user name with which to log into the database.
Password	The password associated with the user name.

The connection information is stored in an xml file whose name consists of the data source name and the suffix `.props`. For example, the properties file for the cdx database is `cdx.props`. You can locate the xml file in the Connections folder.

This is a text file, containing one element for each property. To edit the file, double click on the file name in the Connections folder. The file appears in the editor window. You can make changes to property values as desired.

Additional Programming Considerations

The following sections provide additional information about servlets. For complete information, see the *Programmer's Guide* and the *Foundation Class Reference*.

Handling Servlet Output

Once the servlet has performed all of the associated presentation logic functions, it provides output back to the client. This output contains the results of any business logic activities (content) merged with an HTML or JavaServer Page (layout).

Servlets can invoke JSPs to stream output back to the client.

Calling JavaServer Pages

JSPs can be invoked from servlets via the `RequestDispatcher()` interface, as in the following pseudo code example:

```
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/JSPname.jsp");
dispatcher.forward(request, response);
```

The `forward()` method passes control of the interaction to the JSP, which builds the page and returns it to the client via the `response` object. Additionally, JSPs can be called directly from the client by using a URL

Chapter 6

Creating Presentation Layout

This chapter describes how to create the pages that make up your application user interface.

The following topics are described in this chapter:

- About Presentation Layout
- Adding HTML Pages
- Displaying Pages Using Different Views
- Editing Text with the Format Toolbar
- Adding Components with the Palette
- Adding Objects by Dragging and Dropping
- Creating Hyperlinks
- Creating Forms
- Adding Images
- Inserting Calls to Servlets or Enterprise JavaBeans
- Validating Data Entered on a Form
- Binding JavaServer Page Components to Dynamic Data
- Using RDBM Tag Library

About Presentation Layout

Application developers control the presentation layout of an application with HTML pages and JavaServer Pages (JSPs).

HTML Pages are files containing information to be displayed on a browser. Pages can include graphics, tables of data, and input forms, depending on the designed look and feel of your application.

Some pages are static and unchanging, but some are produced dynamically using JavaServer Pages, and servlets.

A **JavaServer Page (JSP)** is an HTML file, with Sun-specified JSP 1.1 compliant markup that is executed on the web or application server instead of on the client browser. It dynamically creates pure HTML, which is then sent to the client. The JSP has some new keywords—extensions found only in JSPs—which allow the author to embed scripts and expressions in JSP 1.1 scripting language; incorporate other JSPs from the server (conditionally or unconditionally); call Java Servlets; call Enterprise Java Beans (EJBs) to perform tasks on the server; generate output to be incorporated into the dynamic HTML stream, and use JSP 1.1 tag extensions.

JSPs are compiled into Java by the application or web server the first time they are used. After that, they run without recompilation. The reason for using JSPs is to isolate all the dynamic aspects of a web page construction on the server and to leave the browser to just perform display operations. The Application Builder is in control of available features; all the client has to be able to do is display HTML and run JavaScripts.

JSPs act as output templates, separating the presentation of dynamic content from the generation of that content. They use HTML to describe page layout elements, for placement of items such as text and graphics on the page, and page content elements, which determine dynamic content by accessing system variables and business objects, such as Enterprise JavaBeans.

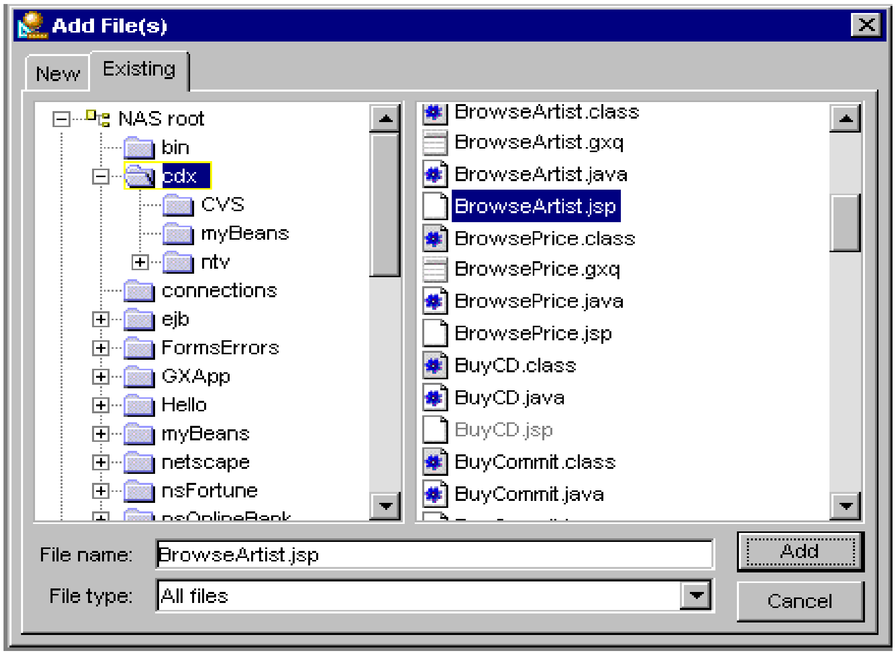
To use JSPs for your application’s presentation layout, you can import existing JSPs and then use the HTML editor text view to edit the embedded Java code.

For more information about Java Server Pages, see the *Programmer’s Guide*.

Adding JSP Files

To add a JSP file, perform the following actions:

1. Choose the Add File(s) menu item from the Project menu.
2. Select All Files for the type file.
3. Locate the file in the iASRoot directory.
4. Click Add to add the file to your project.



File name: BrowseArtist.jsp

File type: All files

Testing and Deploying Pages and JSP Files

Once you have created the required HTML pages and JavaServer Pages for your presentation layout, you can use iPlanet Application Builder to test and deploy them.

Testing HTML Pages and JavaServer Pages

iPlanet Application Builder enables you to test your application's presentation layout by viewing them in a browser.

For more information about how to test HTML pages and JSPs, see "Testing and Deploying Pages and JSP Files."

Deploying HTML Pages and JavaServer Pages

Static pages can be deployed to the web server's HTTP document directory, if there is a web server. You deploy and store JSPs with the other business logic components on the application server.

For more information about deployment, see "Deploying Applications."

Components and the Palette

iPlanet Application Builder provides a number of prebuilt components that you can add to your HTML pages and JSPs to support your application. A component's **properties** control its behavior and appearance. You can customize a component by editing its properties.

Components are organized by type into tabs on a palette. See "Adding Components with the Palette".

Editing Pages and JSPs

You edit static pages and dynamic JSPs using the same methods. The principal difference between them is that static pages are served directly to the browser when called, while JSPs must first be merged with data. For this reason, some components have different properties on JSPs rather than on static pages.

In general, you create pages by running iPlanet Application Builder wizards, and then customize the results by editing existing components and adding new ones. You create JSPs by manually adding the necessary JSP tags to your project files.

Several wizards exist that can help you create application functionality by building pages. For more information, see "Using Wizards to Generate Project Files."

Adding HTML Pages

You can add HTML files to a project in three ways:

- Create a new individual page, as described in "Creating and Managing Projects."
- Add existing pages to the project, as described in "Creating and Managing Projects."
- Use one of the available wizards to create functionality, resulting in both pages and necessary source code.

These wizards are described in detail in "Using Wizards to Generate Project Files."

Displaying Pages Using Different Views

You can view pages from three perspectives:

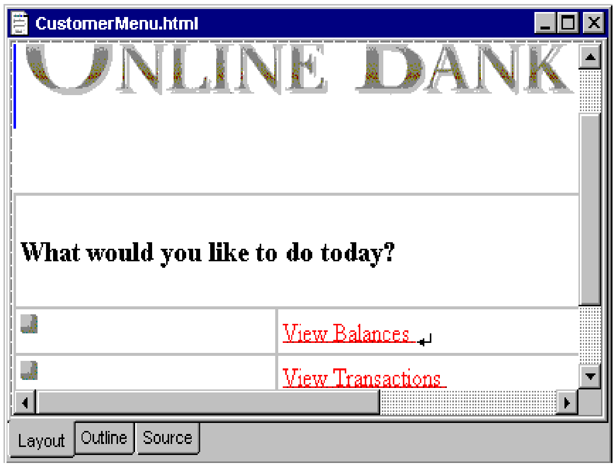
- **Layout view** shows approximately how the finished page looks at runtime, within certain limits.
- **Outline view** displays the page according to its HTML tag structure.
- **Source view** displays the HTML source code that defines your page.

Select a view by clicking one of the tabs at the bottom of the HTML editor window. You can edit items on a page or JavaServer Page (JSP) in any of the three available views.

Using Layout View

Layout view displays the page or JSP in WYSIWYG format, although it may not always appear as the user sees it. Nonvisual components are shown along with the visual components and text. Also, the design-time appearance of dynamic JSP components is often different from their runtime appearance, which can be modified or completely generated by a servlet. For example, a JSP component might show up on screen as `userName`, while at runtime this is replaced with an actual name.

The following image shows the HTML editor window in Layout view:



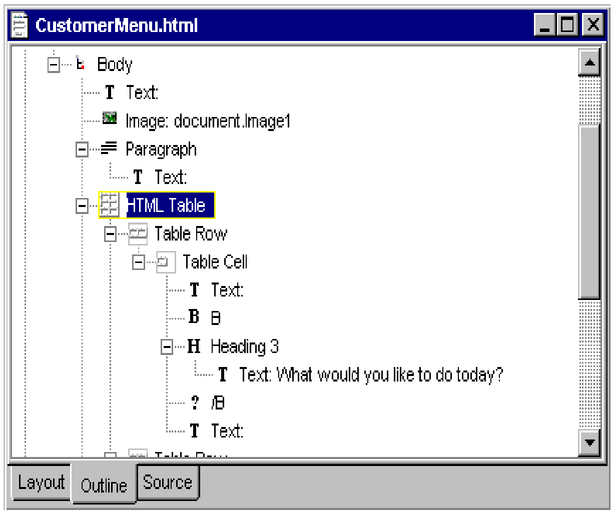
You can add a component to the page by dragging it from the palette into the HTML editor window in Layout view. You can also enter and edit text directly in Layout view at the cursor location. You can insert carriage returns from the keyboard to create new lines, add calls to servlets using the Insert menu, format text using the format toolbar, and drag existing components around the page to change their positions.

Using Outline View

Outline view displays a hierarchical organization of the HTML tags that make up the page. Use this view to work more directly with the page structure.

Selected objects in the outline view also appear in the properties window, where you can examine and change their properties.

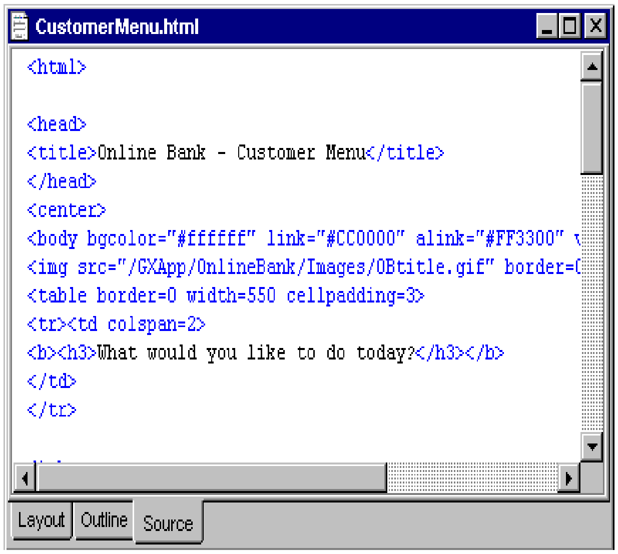
The following image shows the HTML editor window in Outline view:



Using Source View

Source view displays the HTML source code for the page in a text editor window. You can edit the page source directly using this view.

The following image shows the HTML editor window in Source view:



```
<html>

<head>
<title>Online Bank - Customer Menu</title>
</head>
<center>
<body bgcolor="#ffffff" link="#CC0000" alink="#FF3300" v
<img src="/GXApp/OnlineBank/Images/OBtitle.gif" border=(
<table border=0 width=550 cellpadding=3>
<tr><td colspan=2>
<b><h3>What would you like to do today?</h3></b>
</td>
</tr>
```

Some JavaScript code appears on HTML pages between comment tags that signal the begin and end of **generated code**. Do not edit these tags or the code between them, as they are generated and updated by the Editor Beans controlling that component. **Editor Beans** are Java objects that generate and maintain components.

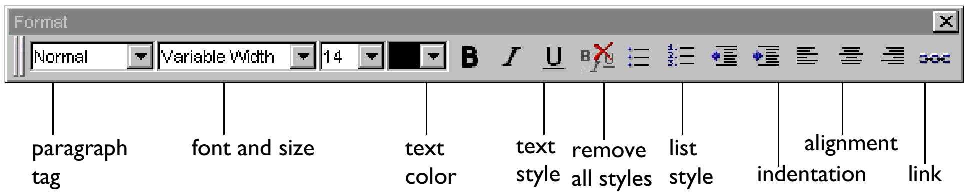
For example, the code shown in bold below should not be edited:

```
<SCRIPT PURPOSE="EVENT_HANDLER"
SCRIPTID="document.Form1.select12.onchange"
LANGUAGE="JAVASCRIPT">
<!--
function document_Form1_select12_onchange() {
alert("user defined event" );
//!! begin generated code !!
if (this.selectedIndex != 0) location =
this.options[this.selectedIndex].value
//!! end generated code !!
}
//-->
</SCRIPT>
```

Additionally, you can use the Source view to embed Java code within JavaServer Pages.

Editing Text with the Format Toolbar

Use the format toolbar, shown below, to edit HTML text in Layout view:



These tools allow you to set the paragraph type, change typeface details, set indentation, and create lists. These tools can also be found on the Format menu. The tools are described in the following table:

Table 6-1 Formatting Tools

Format Tools	Description
paragraph tag	Set the paragraph type for this paragraph. Options include Normal (`<p>`), List Item (`<li>`), and Headings (`<h1>` through `<h6>`).
font type and size	Set the typeface (fixed- or variable-width) and text size.
text color	Set the color for the text foreground. An invisible background is assumed.
text style	Set text style to bold, italic, or underlined. Click to remove all stylistic formatting.
list style	Set list style to bulleted or numbered. Clicking a list style button on an existing list of the same style converts the list to a set of paragraphs.
indentation	Increase or decrease the level of indentation for a paragraph.
paragraph alignment	Align the paragraph to the left, right, or center of the text column.
link	Convert selected text to a hypertext link.
remove all styles	Remove all style attributes and return to default style.

Adding Components with the Palette

Components are reusable objects that appear on an HTML page or JavaServer Page (JSP). Examples include data from a database, images, hyperlinks, form elements such as text entry boxes and Submit buttons. The behavior and appearance of components are defined by their respective **properties**, which can be edited using the properties window. For more information, see “Creating and Managing Projects.”

Components can be Java objects called Editor Beans that generate the required HTML and JavaScript code, depending on the environment into which they are inserted; for example, a text field that generates `<rdbm:field>` tags to data-bind the `VALUE` attribute at runtime. Note that the code generated by Editor Beans should not be edited; see “Using Source View”.

iPlanet Application Builder contains many pre-built components organized on a palette. Many of these components are standard HTML and JSP elements that have been enhanced to include information used by the iPlanet Application Server, while others are standard HTML tags.

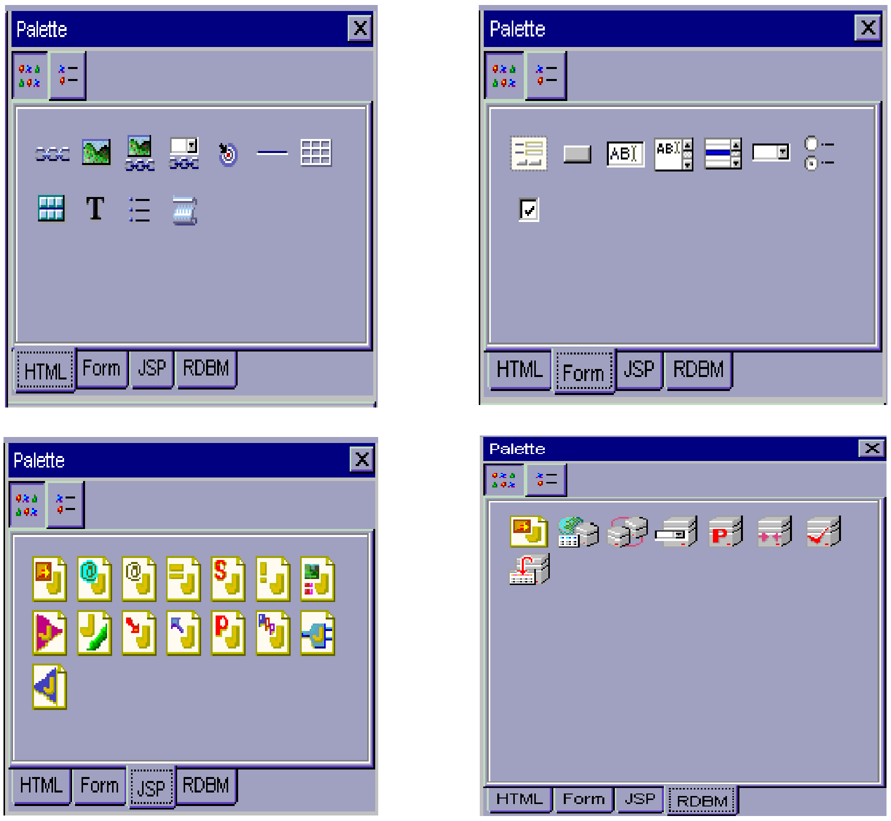
For more information on individual components, see “Component Reference.”

About the Palette

Components reside on the **palette**. Choose Palette from the View menu to launch the palette.

To display a Tooltip that describes the component, move the cursor over the component’s icon on the palette. Click the “List view” button in the upper left corner of the window to display the palette items as a list.

The following images show the palette tabs:



Components are grouped into **palette tabs**, which are shown as short window extensions on the palette. To access components on a given tab, click the tab's name. The following table describes the default palette tabs:

Table 6-2 Default Palette Tabs

Palette Tab	Description
HTML	Components that consist of standard HTML tags, such as Table, Image, and Data-Bound Link. See "HTML tab."
Form	HTML form elements, such as TextField, CheckBox, and PushButton. See "Form tab".
JSP	Components that provide Directives, Expressions, Scriptlets, Declarations, Beans, Properties, Parameters, and plugins.
RDBM	Components for creating RDBM Queries, Loops, Fields, Parameters, Close, Execute, and GoRecord.

Working with Components

To use a component, you add it to your HTML page or JSP by dragging its icon from the palette and dropping it onto the page in Layout view. Note that some data-bound components are only useful in JSPs, and thus can not be dropped onto a static page. For more information on using data-bound components, see “Binding JavaServer Page Components to Dynamic Data”.

You can edit a component’s properties to change its behavior or appearance. For more information, see “Viewing and Editing Project File Properties.”

You can also drag an edited component back onto the palette for later re-use. For more information, see “Customizing the Palette” in “Setting iPlanet Application Builder Options.”

Editing Tags and Components with the Properties Window

Each component and HTML tag behaves according to characteristics defined by the component’s properties. You can edit many of these properties with the **properties window**.

To view or edit the properties of a component or HTML tag, right-click the item in Layout or Outline view and select Properties. The Properties window appears.

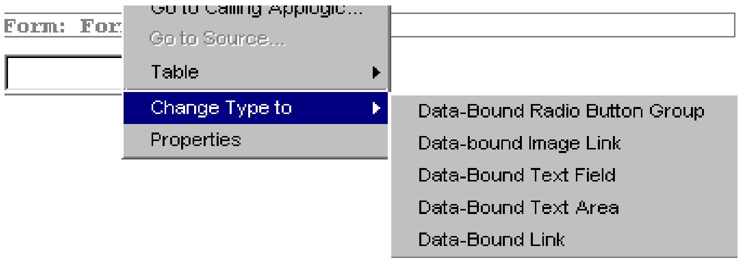
To edit a property, click the property definition on the right. For more information, see “Viewing and Editing Project File Properties” in “Creating and Managing Projects.”

Some properties show a customization button to the right of the property definition box when you edit the definition. This button launches a custom property editor for that property, which helps you specify the property value. For more information, see “Using Custom Property Editors” in “Creating and Managing Projects.”

Converting Component Types

You can convert some components to other, similar components if you want to change their appearance or behavior.

To convert a component, click and hold the right mouse button on the component you wish to convert. The Change Type drop-down option, shown below, lists the component types to which you can convert:



Select the type of component you want. The component converts to the new type.

Adding Objects by Dragging and Dropping

You can move or copy objects from one place to another by dragging them with your mouse. Many objects can be dragged and dropped within the page or from the palette or the Project window. Objects dragged into a page or JSP from outside are copied, while objects dragged within a page are moved.

You can insert a component onto a page by dragging it from the palette and dropping it onto your page (in Layout view) where you want it to appear. The properties window shows the properties for the component.

The following table shows the types of files you can drag from the Project window and drop onto a page or JSP:

Table 6-3 Project Window Drag and Drop File Types

Type of File	**Result when dropped onto an HTML page or JavaServer Page**
Static HTML page	Creates a hypertext link which points to the dropped page.
GIF or JPG image	Inserts the image onto the page in an Image component.
Servlet	Prompts you to choose whether to create a form or a hypertext link to call this servlet. For more information, see “Inserting Calls to Servlets or Enterprise JavaBeans”.

Table 6-3 Project Window Drag and Drop File Types *(Continued)*

Type of File	Result when dropped onto an HTML page or JavaServer Page
JavaServer Page	Prompts you to choose whether to call the JSP as a form or hyperlink, or, if you drop a JSP onto another JSP, to include the dragged JSP's contents using a JSP `include` directive. • Choose to call the dragged JSP using a form or hyperlink if you wish to create a hypertext link to the JSP. • Choose to use a JSP `include` directive if you want the contents of the dragged JSP to be included at transition time.

Creating Hyperlinks

A **hyperlink** is an active section on a page, usually text or an image, which leads to a new page when it is clicked.

You can create text hyperlinks in the following ways:

- Select text and click on the link icon.
- Drag the Data-Bound Link component onto your JSP from the palette and edit its properties.
- Drag an HTML page from the Project window onto your page. A hypertext link is created, with the directory path to the page as the initial text.
- Drag a JSP or servlet from the Project window onto your page. You are prompted for information about the link.

You can create a data-bound text link in the following way:

- Drag the DropDownLink component onto your page from the palette, and edit its properties. This component creates a drop-down box where each item is a hyperlink; when a reader selects an item, the corresponding URL is loaded into the current window or frame.

You can create an image hyperlink in the following way:

- Create an image, select it, then click the Link icon in the toolbar.

You can create a data-bound image hyperlink in the following way:

- First, add the required image to your project (see "Adding Images"). Next, drag the Data-Bound Image Link component onto your page from the palette. Edit its properties, specifying your image as the Image Source.

Creating Forms

You can create forms in the following ways:

Drag any form component from the Form palette tab onto your page. A form is automatically generated around the component. To create a Submit button on the form, drag the PushButton component into the form.

- Drag a JSP or servlet from the Project window onto your page. You are prompted for information about the link. To create a form, choose to link to the servlet with a form.

Adding Images

To add an image to a page or JSP, first copy the image to your web server's root document directory, or to your root project directory. Add the image to your project with the following steps:

1. Choose Add Files from the Project menu or New File from the File menu.
2. Click the Existing tab on the New File(s) dialog box.
3. Locate the image in the file system subdirectory and click Add.
4. Drag the Image or Data-Bound Image Link component from the palette to the location on your page (in Layout view) where you want the image. The component's properties appear in the properties window.

   Alternatively, drag the icon representing the image from the Images folder in the Properties window onto your page. This creates an Image component with this image as the source.
5. Click the Image Source property definition in the properties window. You are prompted to choose an absolute or relative URL pointing to the source for this image. For more information, see "Editing Image URLs" in "Creating and Managing Projects."

Inserting Calls to Servlets or Enterprise JavaBeans

Since JavaServer Pages are compiled into servlets when they are first accessed, you can perform presentation logic tasks with JSPs as well. For more information, see the *Programmer's Guide*.

This section includes the following topics:

- Inserting Calls to Servlets
- Inserting Calls to Enterprise JavaBeans

Inserting Calls to Servlets

To insert a call to an existing servlet you insert a link that calls that servlet. You can insert a call to an existing servlet in four ways:

- Drag the icon representing the servlet file from the Project window and drop it on your HTML page or JavaServer Page (JSP).
- Select Call Existing Servlet from the Insert menu or the right-mouse menu. Choose an existing servlet or browse for one not listed.
- In any hypertext link component (DropDownLink, Data-Bound Image Link, Data-Bound Link), you can specify a servlet as the target of the link. Note that you must provide input parameters if the servlet requires them.
- In any form, you can specify a servlet as the action taken when the Submit button is pressed.

In the first two cases, you are next prompted to choose whether to create a form or a hypertext link to call this servlet. If you choose to create a hypertext link, you are prompted to provide default values for the servlet parameters, and a hypertext link is inserted onto the page.

If you choose to create a form, a basic form is inserted onto the page with appropriate input fields for each of the servlet parameters. The form's Action property automatically refers to the specified servlet. If the file you are calling the servlet from is a JSP, the form components can be populated with data dynamically; see "Binding JavaServer Page Components to Dynamic Data".

If you already have an existing form, you can choose a servlet as the action taken when the Submit button is pressed by setting the form's Action property. Select the Action property, click the custom property editor icon, and then select the servlet you want to call. For more information on this custom property editor, see "Creating and Managing Projects."

You can create a new servlet and insert a call to it all in one step by selecting Call New Servlet from the Insert menu. You are prompted with a series of dialog boxes to create a new servlet and forge the link to it from your page or JSP.

For more information on servlets, see "Creating and Editing Servlets" in "Creating Presentation Logic."

Inserting Calls to Enterprise JavaBeans

iPlanet Application Builder does not provide built-in support to call Enterprise JavaBeans (EJBs) from an HTML page or JSP. However, you can use the EJB API Helper to insert a call in a servlet or within the Java code in a JSP to call an EJB.

Validating Data Entered on a Form

Validation is a runtime check for the validity of user-entered data, such as a social security number or date. The TextField and Text Area components, both found on the Form palette tab, can be validated by a script when text is entered into them. If the user enters inappropriate data, a dialog box appears with a message explaining what data is expected in this field.

Validation is handled by a JavaScript file called `validate.js`. You should never need to edit this file directly.

To set up a TextField or Text Area component for validation, perform the following steps:

1. Select the component's Validation property.
2. Select one of the following validation types from the drop-down list:

Table 6-4 Validation Types

Validation Type	Description	Example of Valid Data
(none)	No validation. This is the default.	anything

Table 6-4 Validation Types

Validation Type	Description	Example of Valid Data
Integer	Positive or negative integer.	-4, 0, 4, 44, 444
PositiveInteger	Integer greater than or equal to zero.	0, 4, 44, 444
Number	Any real number.	-12.5, 0, 12.5, 1000000
Alphabetic	Alphabetic characters A-Z a-z, and spaces.	A man, a plan, a canal
CreditCard	Standard credit card number.	1234 5678 9012 3456
USPhone	10-digit US telephone number.	(123) 456-7890
InternationalPhone	International telephone number.	011 23 4 567 890
Email	Email address, like user@domain.	jefro@haven.com
SSN	A 9-digit social-security number.	123-45-6789
USDate	A date as `mm/dd/yy` or `mm/dd/yyyy`.	12/31/99
IntlDate	A date as `dd/mm/yy` or `dd/mm/yyyy`.	31/12/99
Year	A 2- or 4-digit integer.	99, 1999
Month	An integer between 1 and 12.	1, 2, 3
Day	An integer between 1 and 31.	1, 15, 31
StateCode	2-character US state code.	CA, NY, AK
ZIPCode	US ZIP or ZIP+4 code.	94043, 94043-1234
(Custom)	Custom validation.	

3. If you choose (Custom), you must provide the JavaScript code for your custom validation in the Validation Expression property.

   The expression is evaluated in the context of the field you want to check, thus, you can use unqualified properties in your JavaScript code. Often, you need to use the `value` property, which contains the field's value. For example, the following expression specifies validation of a field so that its values are between 10 and 20:

   ```
   (value >= 10) && (value <= 20)
   ```

   If you want to only accept entries that contain the string 'scape,' as in iPlanet, you could use the following expression:

   ```
   value.indexOf("scape")!=-1
   ```

   The application displays a generic warning message, which is defined in the `validate.js` file, if the user enters invalid data. To change this error message, edit the file and change the contents of the `iCustom` variable.

Binding JavaServer Page Components to Dynamic Data

Dynamic data is represented by merging a JavaServer Page with data obtained by a servlet. JSPs contain placeholders for the incoming data. These placeholders are called **RDBM field tags**, and the result of a data-accumulation activity, such as a database query, is called a result set. The process by which RDBM field tags are connected with the data returned via ResultSets is called **data binding**.

The use of any RDBM tag requires a JSP TagLib directive which refers to the RDBM TagLib.

```
<%@taglib uri=”RDBMS” prefix=”rdbm” %>
```

About RDBM Field Tags

RDBM field tags appear as HTML tags on JSPs. They are replaced with the data they describe at runtime. RDBM field tags look like this in the source view:

```
<rdbm:field name=”field_name” query=”query_name”></rdbm:field>
```

RDBM field tags have a `type` attribute that determines what action the tag performs. The following values for `type` determine the tag type:

Table 6-5 RDBM Tag Types

RDBM Tag Type	Description
`field`	Displays a single dynamic value determined by the `rdbm field` tag, which shows the `ResultSet` and `DataField` to display. See "Displaying a Single Field of Data".
`loop`	Repeats the text block for each row in the `ResultSet`, or a fixed number of times. This tag usually contains one or more RDBM field tags. See"Displaying Multiple Rows of Data".

For more information, see "Using RDBM Tag Library".

About Row Sets

A servlet or JSP can instantiate a row set. Data-bound components in JSPs can be bound to a row set that is instantiated by a servlet.

There are four types of objects you can use as a result set for the purpose of populating a data-bound component:

Table 6-6 ResultSet Types

`ResultSet` type	Description
`DBRowSet`	A result set from one or more queries to a relational database.
`memRowSet`	A result set stored in memory, which does not represent a disk structure such as a row in a table.
`session`	A standard object (defined in `appResource.java`) comprising saved session variables.
`valIn`	A data structure that defines the input parameters to an servlet.

About Data-Binding Properties

Data-binding properties determine which data are displayed in a dynamic component. The following components contain data-bound properties that can be configured to represent data from a servlet for display:

Table 6-7 Components Containing Data-bound Properties

Configured via	Component Type
HTML tab	Data-Bound Image Link, Field, Data-Bound Link, List, Loop
Form tab	CheckBox, Drop-Down List, List Box, Radio Button Group, TextField, Text Area
RDBM	Query, Loop, Field, Parameter, Close, Execute, GoRecord.

There are three standard **data-bound properties** that together make up a data-bound object, linking a single column of data to a component. The standard data-bound properties are listed in the following table:

Table 6-8 Standard Data-bound Properties

Property	Description
`ResultSet`	The result set to be displayed.
`DataField`	The specific column of data in the `ResultSet` to display. This list is determined by the `ResultSet` property; only valid fields in the listed `ResultSet` are shown.
Display Format	An optional **format mask** that reformats raw data at runtime.

The intersection of one or more data sets with one or more data fields, with optional display formats for each, is called a **data expression**. You can use a custom data expression to provide multiple combinations of `ResultSet`, `DataField`, and display format inside one component. To put it another way, the combination of a `ResultSet` and `DataField` (with optional format) is a data expression with only one value.

The data-binding part of a data expression is identified by the notation *`ResultSet.DataField`*. For example, `valIn.LastName` identifies the `ResultSet` “`valIn`” and the `DataField` “`LastName`”.

Output formatting is ignored for `valIn`, `valOut`, and session variables.

The Drop-Down List, List Box, and Radio Button Group components contain a data expression property, which overrides the `DataField` and `Display Format` properties if the `ResultSet` property is blank:

Table 6-9 Data Expression Property

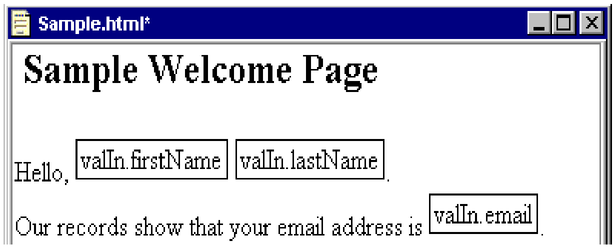
Property	Description
`Label Expression` or `Item Expression`	An expression that organizes multiple combinations of `ResultSet`, `DataField`, and `Display Format` with arbitrary static text so that multiple fields can be displayed in one component.

Displaying a Single Field of Data

A single field of data can be displayed with a `rdbm field` tag. The tag's `query` attribute specifies the `ResultSet` and `name` that identify the field to display.

The Label component defines a single `rdbm field` tag with its `ResultSet`, `name`, and optional `Display Format` properties. Other components that use a single `rdbm field` tag are: Data-Bound Image Link, Data-Bound Link, CheckBox, TextField, and Text Area.

To display multiple fields of data from the same row, use multiple instances of the same component. For example, to display the fields `firstName`, `lastName`, and `email`, all from the `valIn` result set, you can use three different Label components in a row, as in the following sample from Layout view:



Sample Welcome Page

Hello, valIn.firstName valIn.lastName.

Our records show that your email address is valIn.email.

Displaying Single Columns of Data

You can associate a component with a single field of dynamic data by performing the following steps:

1. Select the component you want to associate with dynamic data.

   If the data occurs on multiple rows (in other words, if you are populating a multiple-item component where the items come from the same field of multiple rows in a ResultSet), this component should be set in a loop. See "Using the Loop Component to Display Rows of Data."

2. Select the ResultSet property in the Properties window.

3. Click the custom property editor button and select a ResultSet.

4. Note that some components have more than one result set property, as in the List Box component where the Options result set gathers all the options to be displayed and the optional Selection result set specifies the one to be the initial selection.

5. Click OK to return to the Properties window.

6. Choose a `DataField` object for the current data set by selecting a field from the drop-down box in the `DataField` property.

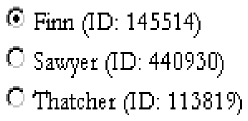
   The Drop-Down List, List Box, and Radio Button Group components have more than one possible `DataField`. The Options `ResultSet` offers both a Labels `DataField`, from which you select the label to be displayed on the page, and an Items `DataField`, from which you select the value to be submitted with the form. These data correspond because they are chosen from the same row in the data set. Follow this step for each `DataField` property.

7. To choose a `DataField` for display, click the `DataField` property definition and select an item from the drop-down box.

8. If you want to apply a format mask to the field, adjust the format mask by selecting the `Display Format` property definition and clicking the custom property editor icon.

Displaying Multiple Columns of Data

You can associate the Drop-Down List, List Box, or Radio Button Group components with multiple dynamic fields, even from multiple result sets. For example, you might want a set of radio buttons to show two fields from a data set, such as a name and an ID number, like this:



You must construct a data expression, because this construction is not possible by dynamically selecting only a single field using the `ResultSet` and `DataField` properties.

To create a data expression, select the `Label Expression` or `Item Expression` property and click the custom property editor button.

Note that complex data expressions can be used to populate iterative lists, as with the single-field data expressions described in "Displaying Single Columns of Data".

Displaying Multiple Rows of Data

Multiple rows of data can be displayed by using an RDBM loop tag as a container for other tags that display each field. The loop tag repeats its text block for each row in the specified data set, or, if specified, a maximum number of times and / or strings with a specified row. The tag's `query` attribute specifies the data set over which the tag's contents iterate.

The **Loop component** defines a single RDBM loop tag. Other components can be inserted into a loop component to display the data. Several other components enable you to display one or more fields of data iteratively, including List, Drop-Down List, List Box, and Radio Button Group.

Using the Loop Component to Display Rows of Data

The Loop component defines a single RDBM `loop` tag with its `ResultSet`, and `Max Rows`. The loop repeats itself for each row of data, up to a maximum number of rows determined by the `Max Rows` property (which corresponds to the `max` attribute).

Using Loops as Table Rows

The Loop component defines a RDBM loop tag but does not create an HTML table. you can insert the loop tag as a row in an HTML table to make the loop a table.

You can also insert a RDBM loop tag as a row in an HTML table as follows:

1. Right-click on a table row and choose the Table drop-down menu.
2. Choose Add Loop to make this table row a loop.
3. You can then insert other components into the loop.

   The contents of the loop repeat according to the loop's properties.

For example, if you have a `ResultSet` called `states` with a `DataField` called `capital`, you could display its contents in a table row with the following steps:

1. Drag the Table component onto your JSP.
2. Select a row in the table and click the right mouse button.
3. Choose Loop and then choose Add Row Loop from the sub-menu.

   The row becomes a loop.
4. Set the loop's `ResultSet (query)` property to `states`.
5. Drag the Label component into the new loop.
6. Set the Label's `ResultSet` property to `states`, and the `DataField` property to `capital`.

   The loop creates a new table row at runtime for each row of data in `states`, and the Label component displays the `capital` field for each row.

To remove a loop, select the row containing the loop you want to remove, click the right mouse button, and choose Delete Loop from the Table submenu.

Using Multiple Loops in a JSP

If you have more than one explicit loop in a JSP, you must edit your JSP to reset the result set's cursor to 1 before using your loops.

For example, if you have a loop in your JSP and then add a data bound list, combo box, drop down list, or radio button component before your loop and the component uses the same result set as your loop, you must reset the cursor.

You can reset the cursor by using the RDBM `goRecord` tag as follows:

```
<rdbm:goRecord query=”your_rowset” start=”1”/>
```

Specifying Custom Formatting for a JSP Field

Often you need to specially format a data field on a JavaServer Page. Although iPlanet Application Server provides a set of formatting masks that you can use to format your output, you may need to do more complex formatting that is not supported directly by the server. For example, if you are displaying the account transactions of a particular customer you might choose to display the transactions with negative values in red.

You can write custom Java code to format data that is returned by result sets for presentation in JSPs. To do so, perform the following steps:

1. Open the JSP with the field you want to format.
2. Select the field you want to format.
3. From the Properties window, open the property editor dialog box for the Display Format property.
4. Select the Custom option for formatting.
5. Specify the name of the Java method that will be used to format the data field.

   You must use a method that is implemented by a `RowSet` subclass. By default, iPlanet Application Server will generate code for a new `RowSet` method with the name you specified in this step. Alternatively, you can choose from a list of methods that currently exist in your servlet.

Using RDBM Tag Library

The query family of tags support declarative declarations of RowSets in a JSP page, along with loops for looping through result-sets and a display tag for displaying column values.

NOTE The use of any RDBM tag requires a JSP TagLib directive which refers to the RDBM TagLib.

1.
   ```
   <rdbm:useQuery id="export_name"
   scope="[page|request|session|application]" command="select *
   from..." queryFile="foo.gxq" queryName="firstQuery"
   execute="[true|false]" dataSourceName="jdbc/..."
   url="odbc:...">...</rdbm:useQuery>
   ```

2.
   ```
   <rdbm:param query="query-declaration-export-name"
   name="name-of-parameter" value="value" bindOnLoad="[true|false]"
   type="[String|Int|Double|Float|BigDecimal|Date|Boolean|Time|Time
   stamp" format="java-format-string for dates">value</rdbm:param>
   ```

3. `<rdbm:loop id="export_name" scope="[page|request|session|application]" query="query-declaration-export-name" start="[request-parameter-name|request-attribute-name|last|constant]" max="integer-maximum-number-of-rows"execute='{true|false]">...</rdbm:loop>`

4. `<rdbm:field query="query-declaration-export-name" name="field name" format="format for doubles" urlEncode="{false/true}">default value</rdbm:field>`

5. `<rdbm:close resource="query-declaration-export-name"/>`

6. `<rdbm:execute query="query-declaration-export-name"/>`

7. `<rdbm:goRecord query="query-declaration-export-name" execute="{false/true}"start="[request-parameter-name|request-attribute-name|last|constant]">default start</rdbm:goRecord>`

useQuery Tag

The `useQuery` tag declares a use of a result-set. It instantiates a `TYPE_SCROLL_INSENSITIVE RowSet`. The `useQuery` tag allows the tool to understand what query is being made, and thereby, what fields are available.

If the result set is already in the scope the `useQuery` wants to save to, the body of the tag is skipped. Although the `RowSet` is created, nothing is done with the created `RowSet`. Some performance gain is seen by lazily creating the `RowSet` and saving the command and connection until the tag knows the `RowSet` is needed.

If the result-set does not already exist, the created `RowSet` is exported using the `id` attribute of the `useQuery` tag at the specified scope, which defaults to request.

Either the command specified is used, or a query located in the `queryFile` is loaded. The name of the loaded query is either the name located in the `queryName` attribute, or, if it is unspecified, the value of the tag's `id` attribute.

Once the rowset is initialized, it may be executed if the execute tag is specified. You must execute a query if you want to use the field tag outside of a loop. This displays only the first record in any query.

The two attributes `queryFile` and `queryName` work in conjunction. The `queryFile` locates the query file. If the value of the attribute is a relative file specification, then the file is looked up in the `RDBMS.path.query` path. This path should be specified as a System property (on UNIX colon separators, on NT semicolon separators) directories. If this variable is not set, then the iAS-specific `RDBM.path.query` property is used. The file is not located relative to the JSP itself. The query file can be created using the File menu Add Query submen.

Additionally, because `RowSets` have numbered parameters, the `useQuery` tag exports Dictionary mapping parameter names (as specified in the command attribute, or in the query file) to Integer offsets as an attribute in the same scope as the `RowSet` using the value of the `id` attribute prepended to `"._params"` as the name for the attribute. This makes setting parameters after the `RowSet` is declared and before it is executed possible.

Param Tag

The `Param` tag sets a parameter on a `RowSet`. The name of the parameter can either be an index or the actual name of the parameter as saved in the Dictionary. If `bindOnLoad` is `true`, the parameter is set before the command is set on the `RowSet`.

NOTE `bindOnLoad` parameters must exist in the body of the `useQuery` tag before any `non-bindOnLoad` parameters.

The value of the parameter is either the value stored in the value attribute, or the contents of the param tag's body. Because JSP1.1 tags don't generally nest (`<%= ... %>` being the notable exception), the only way to bind a parameter to a value from another query is to place a field tag within the body of a parameter tag and have the parameter tag use its body as the value. This is why you can place the value in the body of the tag.

`Param` tags may exist within the `useQuery` tag, in which case they set the parameters directly against their parent query, or elsewhere, probably before a loop tag re-executes the `RowSet`, in which case the parameter is set on the **RowSet** that the `useQuery` tag exported.

Loop Tag

The loop tag loops through the contents of a ResultSet. The query attribute is used to locate the ResultSet or an enclosing useQuery tag. The start attribute is used to indicate the starting position of the loop. Start may either refer to a parameter, or to an attribute, which is looked up using PageContext.findAttribute() or a constant Integer value. The value then indicates which record number to start on, or "last", which will cause the RowSet to scroll to the end, and then back max rows. The max attribute is used to indicate the maximum number of records to display.

Once the loop is finished, the loop tag exports two values -- the number of records that were requested for display (max) and the number of records which were actually displayed. These are saved in attributes using the patterned name id_query_max and id_query_output, respectively. The scope attribute indicates where these attributes are persisted.

If execute is specified, the RowSet is executed before looping begins.

Field Tag

The field tag displays a particular column in the ResultSet. The query attribute locates the enclosing useQuery tag or a previous useQuery tag's exported ResultSet. The name attribute identifies the column name to display. The format attribute allows formating of Strings, numbers, or dates into the appropriate type. The Attribute urlEncode can be used to encode the strings. If the column is null, the body of the field tag is output.

Close Tag

The close tag releases resources back to the system. The resource attribute locates the exported query resource (ResultSet) and calls close() on it.

Execute Tag

The execute tag executes the identified query.

goRecord Tag

The `goRecord` tag optionally executes the specified query and moves the `ResultSet` to the record indicated by the start attribute. Start can refer to a parameter, an attribute, or a constant. If the start attribute is last,the result-set is moved to the last record.

Example

The following tags produce the displayed output.

```
<HTML>
<BODY>
<%@taglib prefix="rdbm"uri=" %>
<h2>Now let us see</h2>
<rdbm:useQuery id="a" queryFile="queries.gxq"
dataSourceName="jdbc/cdx">
</rdbm:useQuery>
<rdbm:useQuery id="b" queryFile="queries.gxq"
dataSourceName="jdbc/cdx">
</rdbm:useQuery>

<table border=1 cellPadding=3>
   <tr>
   <th>name</th>
   <th>phone</th>
   <th>Titles Owned</th>
   </tr>

   <rdbm:loop id="loop1" query="a" max="5" execute="true">

      <tr>
      <td><rdbm:field query="a" name="name"/></td>
      <td><rdbm:field query="a" name="phone"/></td>
      <td><rdbm:param query="b" id="owner" type="Int">
         <rdbm:field query="a" name="id"/></rdbm:param>

         <table border=1 cellPadding=3 width="100%">
         <tr>
         <th>title</th>
         <th>price</th>
         <th>artist</th>
         </tr>

         <rdbm:loop id="loop2" query="b" max="5" execute="true">
```

```
          <tr>
          <td><rdbm:field query="b" name="title"/></td>
          <td><rdbm:field query="b" format="$#,###.00"
             name="price"/></td>
          <td><rdbm:field query="b" name="artist"/></td>
          </tr>

          </rdbm:loop>

</table
</td>
</tr>

</rdbm:loop>

</table>

<rdbm:close resource="a"/>
<rdbm:close resource="b"/>

</BODY>
</HTML>
```

Numeric Format Characters

Table 6-10 Numeric Format Characters

Character	Meaning	Example Data	Example String	Example Result
#	Unfilled digit placeholder. Replaced by numeric digits in the data. If the data to format has fewer digits than the format string, the empty places are not filled. The output can be shorter than the format string.	679.649 700	#####.## $#,###.00	679.64 $700.00
0	Zero-filled digit placeholder. Replaced by numeric digits in the data. If the data to format has fewer digits than the format string, the empty places are filled with zeros. The output is always at least as long as the format string. When placed to right of decimal point, indicates precision. Data is rounded if necessary.	679.649 679.649 700	0000.00 00### $000.00	0679.65 00679 $700.00
,	Thousands separator. A separator character will appear between every three digits to the left of the decimal point in the output data.	1234	0,000	1,234
.	Decimal separator. A decimal point character will appear between the whole and fractional parts of the output data.	679.649 700	000.00 $#,###.00	679.65 $700.00
;	Separates a pair of formats. The first format is used for positive numbers, the second for negative numbers.	23 -66	##;(##) ##;(##)	23 (66)
Literals such as $ + - ( ) and space characters	Any character in the format string that is not a special character will appear in the output data exactly as typed. If using parentheses, always use matched pairs.	5552365 69100 123	###-#### $0,000 0 0 0	555-2365 $69,100 1 2 3

Date/Time Format Characters

Table 6-11 Date/Time Format Characters

Character	Meaning	Example Data	Example String	Example Result
D	Day of the month, with no leading zero.	1/1/2020	M-D-YY	1-1-20
DD	Day of the month, with leading zero.	1/1/2020	MM-DD-YY	01-01-20
DDD	Day of the week, abbreviated.	7/4/1996	DDD	Thu
DDDD	Full day of the week.	7/4/1996	DDDD	Thursday
M	Number of the month, with no leading zero.	1/1/2020	M-D-YY	1-1-20
MM	Number of the month, with leading zero.	1/1/2020	MM-DD-YY	01-01-20
MMM	Name of the month, abbreviated.	1/1/2020	MMM	Jan
MMMM	Full name of the month.	1/1/2020	MMMM	January
Y	Number of the day in the year (1–366).	1/1/2020	Y	1
YY	Last 2 digits of the year.	1/1/2020	M-D-YY	1-1-20
YYY or YYYY	All 4 digits of the year.	1/1/2020	M-D-YYYY	1-1-2020
h	Hour from 1–12, with no leading zero.	8:05 pm	h:mm	8:05
hh	Hour from 1–12, with leading zero.	8:05 pm	hh:mm	08:05
H	Hour from 1–24, with no leading zero.	8:05 pm	H:mm	20:05
HH	Hour from 1–24, with leading zero.	3:00 am	HH:mm	03:00
m	Minute, with no leading zero.	3:09 am	h:m	3:9
mm	Minute, with leading zero.	3:09 am	h:mm	3:09
s	Seconds, with no leading zero.	3:09 am	h:m:s	3:9:0
ss	Seconds, with leading zero.	5:07:02 am	hh:mm:ss	05:07:02
AM/PM	Adds letters AM or PM after the date/time data to indicate morning or afternoon/evening hours.	5:07:02 am	h:mm AM/PM	5:07 AM
Literals such as / - : and space characters	Any character in the format string that is not a special character will appear in the output data exactly as typed.	1/1/2020 8:05 am	m-d-yy h:m:s	1-1-20 8:5:0

Chapter 7

Creating Business Logic

This chapter describes how to create **Enterprise JavaBeans (EJBs)**, which determine the bulk of your application's actual data and rules processing.

EJBs enable you to partition your business logic, rules, and objects into discrete, modular, and scalable units. Each EJB encapsulates one or more application tasks or application objects, including data structures and the methods that operate on them.

Note that you should consider creating your project's EJBs with an external IDE and import them into iPlanet Application Builder.

The following topics are described in this chapter:

- About Enterprise JavaBeans
- Creating Session EJBs
- Creating Entity EJBs
- Importing EJBs
- Examining and Changing EJB Properties

About Enterprise JavaBeans

The Enterprise JavaBeans architecture is a component-based model for development and deployment of object-oriented, distributed, enterprise-wide applications. An **Enterprise JavaBean (EJB)** is a single element in such an application. Applications written using EJBs are scalable, encapsulate transactions, and permit secure multi-user access. These applications can be written once and then deployed on any server platform that supports EJBs.

If servlets act as the central dispatcher for your application and handle presentation logic, EJBs do the bulk of your application's actual data and rules processing. EJBs enable you to partition your business logic, rules, and objects into discrete, modular, and scalable units. Each EJB encapsulates one or more application tasks or application objects, including data structures and the methods that operate on them. Typically, they also take parameters and send back return values.

The fundamental characteristics of EJBs are as follows:

- Creation and management of beans is handled at runtime by a container.
- Customization of beans is handled at deployment time by editing the beans' environment properties.
- Manipulation of metadata, such as transaction mode and security attributes, are separated from the enterprise Bean class, and handled instead by the container's tools at design and deployment time. Note that you actually manipulate the bean that runs in the container, and not the container. One of the key features of an EJB is that you can specify transaction management, isolation level, and security models on a per-bean basis. However, you can also define these features at deployment and runtimes.
- Mediation of client access is handled by the container and the server where the bean is deployed, freeing the bean designer from having to process it.
- Restricting a bean to use standard container services defined by the EJB specification guarantees that the bean can be deployed in any EJB-compliant container.
- Extending a bean to use specific container features prevents a bean from being deployed in other containers that do not support those features. It is possible to develop an EJB that can determine its environment and run appropriately.
- Including a bean in, or adding a bean to an application made up of other, separate bean elements—a "composite" application—does not require source code changes or recompiling of the bean.
- Definition of a client's view of a bean is controlled entirely by the bean developer. The view is not affected by the container in which the bean runs or the server where the bean is deployed.

Types of EJBs

There are two kinds of EJBs: session and entity. Each of these bean types has different uses in a server application. An EJB can be an object that represents

- a stateless service
- a session with a particular client, which automatically maintains state across multiple client-invoked methods
- a persistent entity shared among multiple clients.

Session EJBs

A session bean typically:

- Executes on a single client.
- Handles transaction management according to property settings, which includes custom transaction management.
- Updates shared data in an underlying database.
- Provides only indirect access to databases.
- Is relatively short lived.
- Is destroyed if the EJB server terminates.

A session bean implements its own business logic. All functionality for remote access, security, concurrency, and transactions is implemented by the EJB container. A session EJB is a private resource used only by the client that creates it.

In iAS, an EJB that encapsulates business rules or logic is a session bean. The life duration of a session EJB is usually brief. For example, you might create an EJB to validate user database logins. Each time a user logs into your database, the application creates the session bean to validate the login. Once the login is validated or rejected, the session bean is freed.

Entity EJBs

An entity EJB typically:

- Represents data in the database.
- Is transactional.
- Allows shared access for any number of users.
- Can exist as long as its data in a database.
- Is persistent and exists when the EJB server is restarted.

The server that hosts EJBs and an EJB container provide a scalable runtime environment for many concurrently active entity EJBs. Entity EJBs represent either persistent data (such as a database or document), or an object that is used by an existing enterprise application.

An EJB encapsulates a business object, such as a database connection. The life duration of an entity EJB can span the entire life of the application that instantiates it, or it can span some portion of it. For example, suppose you are in the automotive parts industry. You might create an inventory control entity bean that interfaces with your parts database everywhere throughout your application.

Creating Session EJBs

Session beans are intended to represent temporary objects, such as updating a single database record, a copy of a document for editing, or specialized business objects for individual clients, such as a shopping cart. These objects are available only to a single client. When the client is done with them, the objects are released. When you design an application, designate each such temporary, single-client object as a potential session bean. For example, in an online shopping application, each customer's shopping cart is a temporary object. The cart lasts only as long as the customer is selecting items for purchase. Once the customer is done, and the order is processed, the cart object is no longer needed and is released.

Like an entity bean, a session bean accesses a database through a combination of bean settings for transaction control and JDBC calls. These transaction settings and JDBC calls are refereed by the session bean's container, which is transparent to you as the developer. The container provided with iAS makes uses of the iAS Database Access Engine (DAE) to handle the JDBC calls and result sets.

For information about using the Session EJB wizard to create an EJB, see "Using the Session Bean Wizard".

Creating Entity EJBs

Entity beans are intended to represent persistent objects, such as rows in a database, documents, and specific business objects that may be accessed by multiple clients. When you design an application designate each database, document and business object as a potential entity bean. For example, an inventory control system combines a database with specialized methods for search, retrieval, removal, restocking, and reporting. The inventory control system is a perfect candidate for an entity bean.

An entity bean accesses a database through a combination of bean settings for transaction control and JDBC calls. These transaction settings and JDBC calls are refereed by the entity bean's container, which is transparent to you as the developer. The container provided with iAS makes use of the iAS Database Access Engine (DAE) to handle the JDBC calls and result sets.

The high-level ability to support transactions through EJB properties is especially useful for entity beans because the bean can encapsulate control for distributed transactions that span multiple data sources without requiring you to handle the details.

While an entity bean may represent the database to one or many clients, often there are specific client database events that are unique to the client. In these cases, you may want to use a session bean to control a client's interaction with the entity bean's database representation.

For information about using the Entity EJB wizard to create an EJB, see "Using the Entity Bean Wizard".

Importing EJBs

To import an EJB, perform the following steps:

1. Copy the JAR file to a directory under the iAS root.
2. Select Import EJB(s) from the Project menu.
3. Select the JAR file, either by navigating or by entering the file name, and click Add.

The iPlanet Application Builder unpacks the JAR file and places the files it contains into the appropriate project folders. The JAR file itself is not modified not is it imported into the project.

You can use the Add File(s) menu item in the Project menu to bring non-EJB JAR files into a project. These files are placed in the project's Miscellaneous Files folder.

Examining and Changing EJB Properties

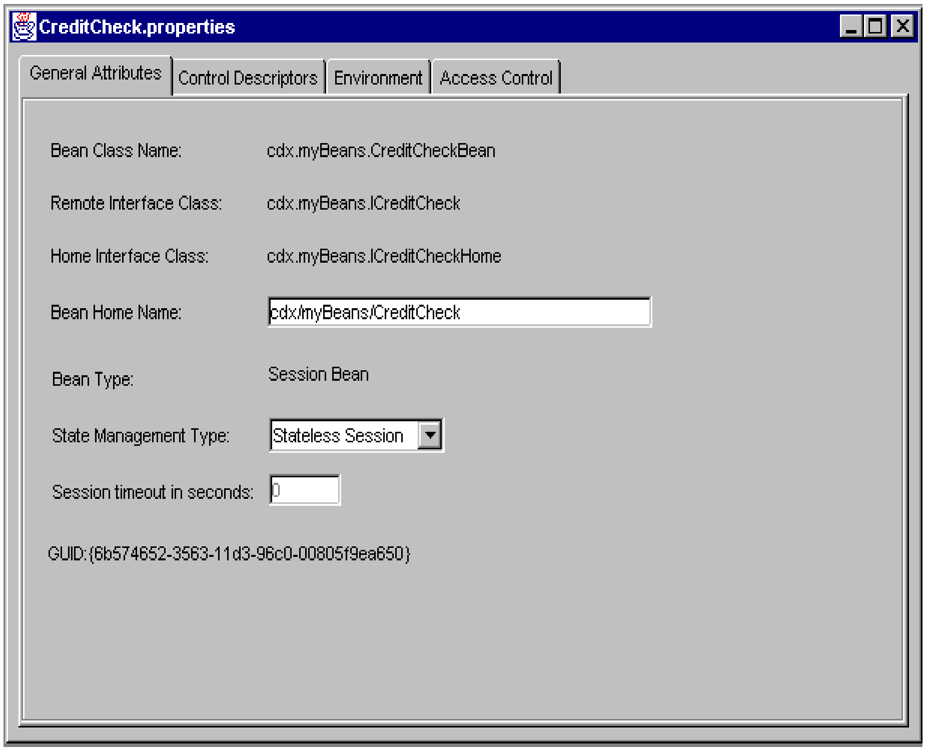
When you create an EJB, iPlanet Application Builder creates a `.properties` file for it. This file contains meta-information for all other files in the EJB. You can double-click on this file's icon in the Project window to open a Properties editor that allows you to view and change this information.

EJB properties are displayed in four panels:

- General attributes
- Control descriptors
- Environment
- Access control

Many of these properties are set when you create the EJB using a wizard. For information for the data you specify in the Session and Entity EJB wizards, see “Using Wizards to Generate Project Files”. For information about Enterprise JavaBeans and how you can create XML files using the iPlanet Application Server, see the *Programmer’s Guide*.

The general attributes panel contains information about the class names and the kind of bean.



Bean Class Name: cdx.myBeans.CreditCheckBean

Remote Interface Class: cdx.myBeans.ICreditCheck

Home Interface Class: cdx.myBeans.ICreditCheckHome

Bean Home Name: cdx/myBeans/CreditCheck

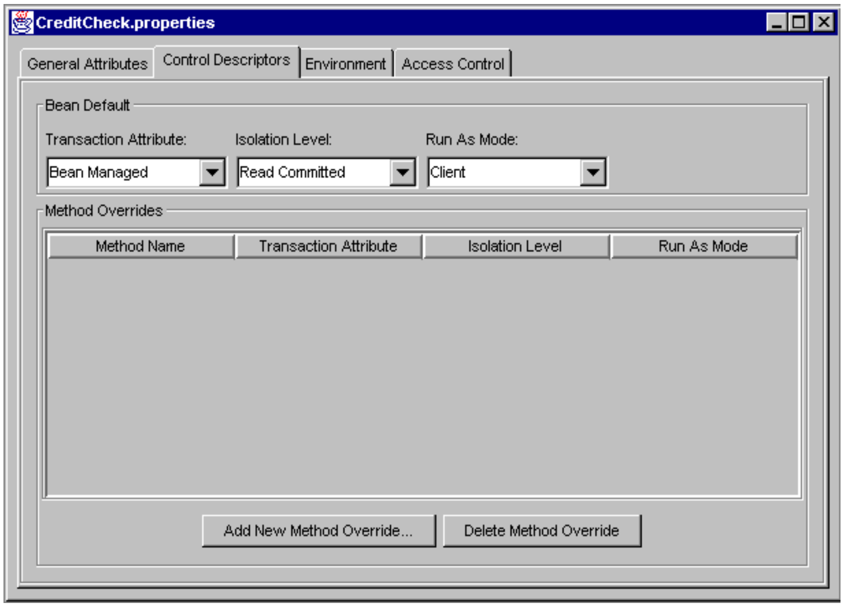
Bean Type: Session Bean

State Management Type: Stateless Session

Session timeout in seconds: 0

GUID:{6b574652-3563-11d3-96c0-00805f9ea650}

The control descriptors panel contains information about the transaction attribute, isolation level, and run mode. You can change these descriptors if you want. You can also add or delete methods that override those defined in the bean.

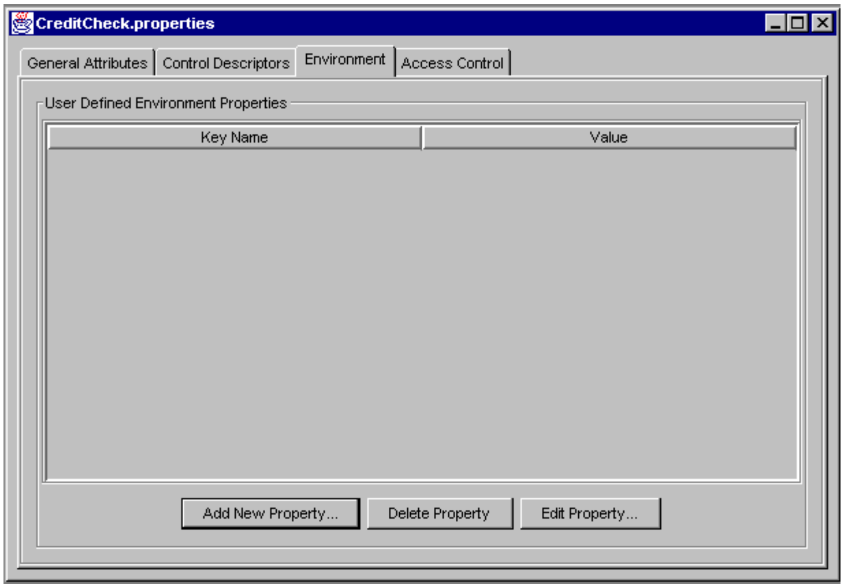


Transaction Attribute: Bean Managed
Isolation Level: Read Committed
Run As Mode: Client

Method Overrides

Method Name	Transaction Attribute	Isolation Level	Run As Mode

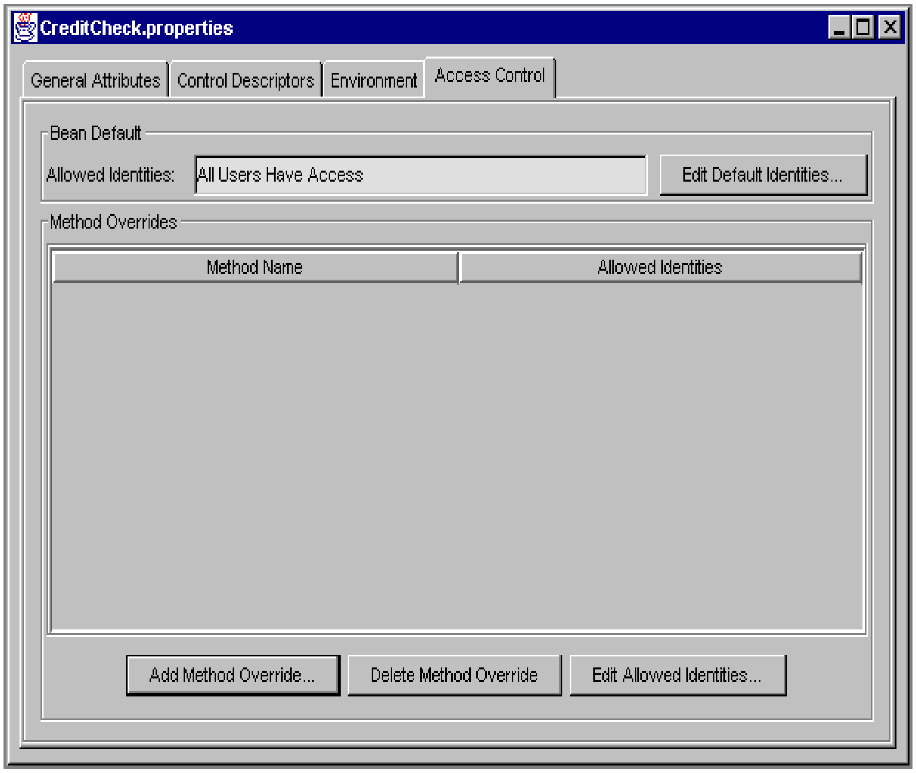
The environment panel contains a list of keys and their values. You can modify them if you want.



User Defined Environment Properties

Key Name	Value

The access control panel specifies the default access control and allows you to specify access for specific methods.



Allowed Identities: All Users Have Access

Chapter 8

Using Wizards to Generate Project Files

This chapter describes the iPlanet Application Builder code generation wizards that help you perform common tasks associated with building an iPlanet application.

The wizards provide a quick method for automatically generating and linking together files needed for advanced application features. The following topics are described in this chapter:

- About iAB Wizards
- Using the Data Model Wizard
- Using the Input Wizard
- Using the Results Wizard
- Using the Database Forms Wizard
- Using the Query Wizard
- Using the HTML Page Wizard
- Using the JSP Wizard
- Using the Servlet Wizard
- Using the Java File Wizard
- Using the Session Bean Wizard
- Using the Entity Bean Wizard

About iAB Wizards

iPlanet Application Builder provides easy-to-use code generation wizards to create sets of HTML and JavaServer pages, servlets, and database queries. To use these wizards most effectively, it is best to think of your application in terms of activities, and from there, map each activity to the use of one or more Wizards.

The tasks common to most applications involve interactions between application users and the Servlets and JSPs that make up the application's business logic. iPlanet Application Builder provides two generic wizards to help you create the files necessary for gathering input from a user (the Input wizard) and giving information to the user by way of an HTML template (the Results wizards). There is also one wizard for generating input forms and output reports from a relational database (the Database Forms wizard), and one for implementing a login page with optional password validation (the Login wizard).

Wizards are invoked from the New File(s) dialog box. The following table describes the available wizards, shows their icons, and describes the types of files generated by each wizard (note that you can choose from existing files rather than generate new ones):

Table 8-1 Application Builder Wizards

Icon	Wizard	Description	Generated Files
	Data Model	Creates a data model.	• Data model
	Input	Creates a generic input form and a servlet to handle the input.	• HTML page with a data input form • Servlet to process the form, optionally accessing a query • Empty JSP to display results • Insert query to store data to a data source (optional)
	Results	Creates a generic servlet and JSP	• Generic servlet linked to the JSP • Empty JSP
	DatabaseForms	Enables the user to perform interactions with a relational database and generate complex reports and forms.	• HTML page with a data input or query form and optional search form • Query to retrieve data from database • Servlets to process the query • JSP(s) to display the results of the query

Table 8-1 Application Builder Wizards *(Continued)*

Login	Creates login functionality, optionally with password verification.	• HTML page with a login form • Servlet to process the form • JSP to use as a results page for successful login
Query	Creates a SQL statement.	• Query to perform the SQL operation.
HTML Page	Creates an HTML page.	• Empty HTML page
Java-Server Page	Creates a JavaServer page.	• Empty JavaServer page
Servlet	Creates a servlet.	• Java code file
Java File	Creates a java class.	• Empty Java code file
Session Bean	Creates a session bean.	• home interface source file • remote interface source file • enterprise-bean class source file • `<projectname>.xml` files
Entity Bean	Creates an entity bean.	• home interface source file • remote interface source file • enterprise-bean class source file • `.xml` files

In all of these wizards, click Next to move on to the next step, or click Back to return to a previous step.

Using the Data Model Wizard

The data model wizard allows you to specify the database tables and relationships between them. You must create a data model before you can create a query or form. The following files are generated, with filenames based on the name you specify:

Table 8-2 Data Model Wizard Generated Files

Generated File	File Name
Data model definition	`filename.kdm`

To launch the Data Model wizard, select New File(s) from the Project menu. Select Data Model and click Add. The Insert Tables dialog appears, which allows you to use an existing connection or create a new one. Previously established connections are defined as a *.xml file in iAS's Connections folder. If you create a new connection, you must specify the driver, data source, username, password, and, optionally, a database name, for your connection.

After choosing or creating the connection, the Add Tables dialog appears, allowing you to specify the tables you want to include in your data model. Once the tables have been identified, you complete the data model by defining relationships between fields in the tables. iPlanet Application Builder displays a visual representation of the tables from which you select a field from one table and drag it to its related field in another table. By default, each relationship you create is one-to-many from the dragged field to the destination field. You can change the kind of relationship by selecting the line between the fields and modifying the Relationship property in the Properties editor.

When you finish defining relationships and close the window, iPlanet Application Builder prompts you for the file name in which to save your data model.

Using the Input Wizard

The Input wizard creates a generic input form and a servlet to handle the input. The following files are generated, with filenames based on the name you specify:

Table 8-3 Input Wizard Generated Files

Generated File	File Name
HTML page with a data input form	`filename.html`
Servlet to process the form	`Calledfilename.java`
Empty HTML template to display the results	`Calledfilename.jsp`
Insert query to store data to a data source (optional)	`Calledfilename.gxq`

The generated HTML form consists of a basic form structure containing the input parameters you specify in the wizard. The form's SUBMIT button submits the form to the generated servlet, which processes the input and merges results with the generated JSP.

1. To launch the Input wizard, select New File(s) from the Project menu.
2. Select Input and click Add.

   The Specify Page Name and Location dialog box appears.
3. Enter a base filename and location for the generated files.
4. Choose how you would like to process the input information, depending on how the input parameters are processed.

   You can choose Handle Information with Custom Code or Store to a Database Table, described in the following sections.

Handle Information with Custom Code

If you choose to handle the input with custom code, the wizard prompts you for the input values that you want to make available to your servlet. Choosing to use custom code causes the wizard not to generate a query file.

Store to a Database Table

To enable the input to be inserted directly into a database using a new INSERT query, choose "Store to a database table" and click Next.

Perform the following steps to store the input to a database:

1. First, select a data model on the Specify Data Model dialog box.

   If you have not yet created a data model for this project, or if your data model is not appropriate for this input, select Create a New Data Model to exit the Input wizard and create a new data model.
2. In the Specify Data Connection dialog box, choose "Use an existing connection".

   If you have already set up a database connection. Otherwise, choose "Create a new connection" and click Next. Choose a database driver, a data source, and a user name and password.
3. In the Select Input Fields dialog box, choose a table from your data model.

   The fields in the table appear in the left column. For each table, choose whether the field appears on the form at runtime.

You must edit the generated JSP to display the results.

Using the Results Wizard

The Results wizard creates a generic servlet linked to an empty JavaServer Page (JSP). You can then create presentation logic in the JSP and transaction logic in the servlet. The following files are generated, with filenames based on the name you specify:

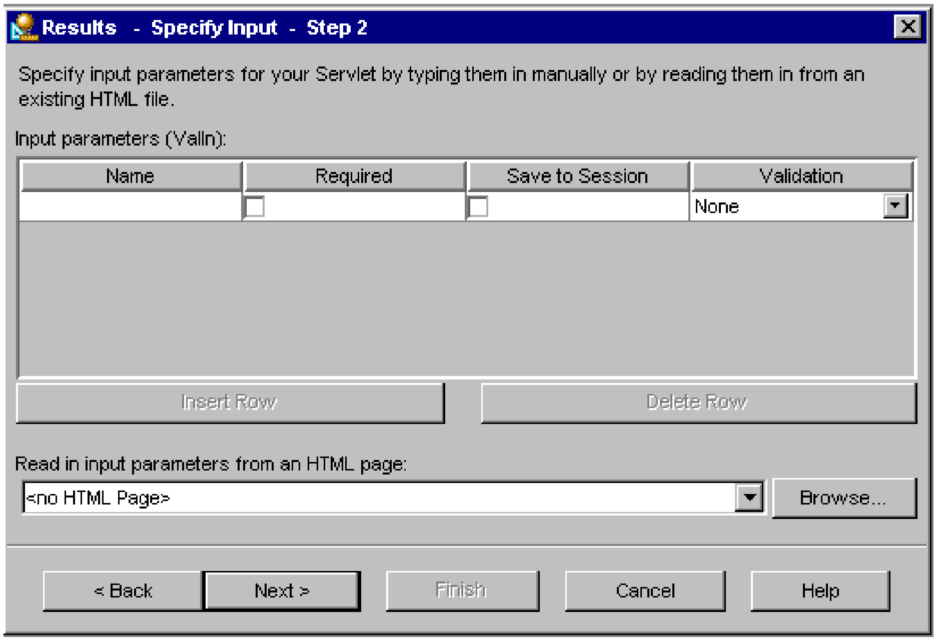
Table 8-4 Results Wizard Generated Files

Generated File	**File Name**
Empty JavaServer Page	`filename.jsp`
Generic servlet linked to the JSP	`filename.java`

The generated servlet is a generic stub you can use to develop a piece of your application. The results of the servlet activities merge with the generated JSP to produce an HTML page at runtime.

To launch the Results wizard, select New File(s) from the Project menu. Select Results and click Add. The Specify Servlet Name and Location dialog box appears. Enter a base filename and location for the generated files.

Specify input parameters for the servlet. If you already have an HTML input page, you can read input parameters from it automatically. Alternatively, you can enter the parameters by hand using the Specify Input dialog box, shown here:


Results - Specify Input - Step 2

Specify input parameters for your Servlet by typing them in manually or by reading them in from an existing HTML file.

Input parameters (ValIn):

Name	Required	Save to Session	Validation
	☐	☐	None

Read in input parameters from an HTML page:

<no HTML Page>

To read parameters from an existing page, select the page from the drop-down box at the bottom of the Specify Input dialog box, or click Browse to search the filesystem for the page. To specify parameters by hand, enter the name and select the appropriate settings for other properties.

Using the Database Forms Wizard

The Database Forms wizard enables you to set up interactions with a relational database and generate complex reports. The following files are generated, with filenames based on the name you specify:

Table 8-5 Database Forms Wizard Generated Files

Generated File	File Name
HTML page with a data input form and optional search functionality	`filename_Search.html`
Optional query file to retrieve data from database	`filename.gxq`
Servlets to process the transaction query	`filename.java`

Table 8-5 Database Forms Wizard Generated Files

Generated File	File Name
JavaServer Page to display the results	`filename.jsp`
Servlets to process the detail transaction query (optional)	`filenameDetail.java`
JavaServer Page to display the detail results (optional)	`filenameDetail.jsp`

The generated servlet processes form for transaction (such as update, insert, delete,or search data from generated HTML files). JSP extended taglibs use the data model and query file to retrieve data from a relational database, and merges this data with the generated JSP to display the results as a static report or an editable form.

To launch the Database Forms wizard, select New File(s) from the Project menu. Select Database Forms and click Add. The Specify Servlet Name and Location dialog box appears. Enter a base filename and location for the generated files.

Choose how you would like to define the data used on your forms and reports. You can choose Use an Existing Query File, or choose Use a Data Model to create a new query file.

Use an Existing Query File

Choose a query file from the drop-down list, or browse for one. When you find the file you want to use, the Select a Query dialog box prompts you to choose a single query from the queries in the file.

If you choose a `select` query, the Select Records Display dialog box prompts you to choose whether records should be displayed as a single record, in which only the first row of data found by the query appears in the final output, or multiple records, where all rows of data appear in the final output.

Use a Data Model

If you choose to create the details of your query from scratch, use the Select Pages to Create dialog box to choose what kind of output pages you want to produce.

A **master record** is the primary target of a query. A **detail record** is the result of a secondary query, based on the master record. Combinations of master and detail queries are shown. If a set of pages can be produced, they are linked together with standard VCR-type navigation tools (first record, next record, previous record, and last record).

The types of output pages are described below:

Table 8-6 Types of Output Pages

Type of Pages	Example
Single record master	An account
Multiple record master	A list of accounts
Single record master and multiple record detail (single page)	A single accounts and a list of transactions on that account
Single record master and multiple record detail (multiple, linked pages)	A single account and a list of all transactions on that account
Multiple record master and single record detail	A list of accounts, and a detailed view on each account
Multiple record master and multiple record detail	A list of accounts, and a list of all transactions on each account

Additionally, you can check the "Create a search page" checkbox to generate a form that enables users to search for certain records by specifying search criteria.

Specify a data model (or exit the wizard to create one). On the Specify Data Connection dialog box, choose "Use an existing connection" if you have already set up a database connection. Otherwise, choose "Create a new connection" and click Next. You are prompted to choose a database driver, a data source, and a user name and password.

Depending on the type of output you want to produce, you are prompted for display field information, first for the master query and next for the detail query. Complex pages are shown with either a master section (for single-page output) or a master page (for multiple-page output).

In each case, you must provide the following information if it is relevant to your selected output scenario:

Table 8-7 Required Information for Dialog Boxes

Dialog Box	Information to Provide
Select Display Fields	Choose the fields you want to display on the page or section by selecting a table and one or more fields, and moving the fields into or out of the "Fields to display" column using the left and right arrow buttons between columns. Doubled arrows move all fields at once. You can use the up and down arrows on the right to adjust the order of the fields. You can use fields from multiple tables.
Select Table Format	Choose which fields are editable if you are creating a form as output, or choose "Display-only fields" to create a database report.
Select Sort Fields	Choose which field(s) to use as a sort key when displaying multiple records. This dialog box functions similarly to the Select Display Fields dialog box described above, with the addition of buttons you can use to specify the field to be sorted ascending or descending.
Specify Primary Key(s)	Choose a field that guarantees the uniqueness of a row in the query. Note that database-controlled fields (such as auto-increment fields) usually cannot be updated by a normal database transaction; in this case, be sure to specify any auto-increment fields as noneditable when prompted.
Select Drilldown Field	Choose a field to use as a hypertext link from the master page to the detail page by selecting a field from the "Display field" drop-down box. Then specify columns to join, and the operator for joining them. The master fields automatically become links to the appropriate detail pages.
Select Columns for Search Page	If you chose to create a search page, choose which fields in the query should appear on the search page, and what kind of search should be possible.

Using the Login Wizard

The Login wizard enables you to create login functionality, optionally with password verification. The following files are generated, with filenames based on the name you specify:

Table 8-8 Login Wizard Generated Files

Generated File	File Name
Static HTML page with a data input form	`filename.html`
Optional query file to retrieve data from database if login is authenticated by database table	Called`filename.gxq`

Table 8-8 Login Wizard Generated Files

Generated File	File Name
Servlet to process the login	`Calledfilename.java`
JavaServer Page to display the results of a successful login	`Calledfilename.jsp`

To launch the Login wizard, select New File(s) from the Project menu. Select Login and click Add. The Specify Page Name and Location dialog box appears. Enter a base filename and location for the generated files.

Specify input parameters in the Specify Input dialog box. These parameters appear as text input boxes in a form on your login page. The servlet uses these parameters to verify the login. Default input parameters are Username and Password.

For each parameter, enter or choose the following information:

Table 8-9 Parameter Attribute Options

Parameter Attribute	Description
Name	Parameter name
HTML Type	Type of entry box for this parameter. Valid types include the following:
	Text (plain text on a single line)
	TextArea (plain text on multiple lines)
	Password (entry replaced on user's screen with asterisks)
	Checkbox (checked yields `true`, unchecked yields `false`)
Default Value	The value submitted with the form if the user enters nothing.
Validation	Optional validation for Text, TextArea, or Password fields.
Required	Check this box if this parameter is required.

The Select Login Authentication dialog box prompts you to choose a method for verifying that the provided parameters constitute a valid login. The available options, Database Table, LDAP Server, and Custom, are described in the following sections.

Database Table

To authenticate your login against a database, first choose a data model, or exit the wizard and create one if needed. Create a database connection by providing a database driver, a data source, and a user name and password.

On the Specify Database Login Information dialog box, choose a database table that contains login fields, and then match your input parameters with database fields.

You can save any of the fields in the table as session variables by checking the Save to Session checkbox for that field on the Specify Query Outputs to Save to Session dialog box.

LDAP Server

The Specify LDAP Server Information dialog box prompts you for information to enable a connection to the LDAP server to authenticate the login. Specify the same server name, port, organization and country as you did when you installed your directory server. Use the drop down list to choose the source of the values for the name and password.

Custom

For Custom authentication, you must provide code in the generated servlets to authenticate your users.

Using the Query Wizard

The Query wizard creates a SQL query from the tables in your data model. The following files are generated, with filenames based on the name you specify:

Table 8-10 Query Wizard Generated Files

Generated File	File Name
Query file	`name.gxq`

To launch the Query wizard, select New File(s) from the Project menu. Select Query and click Add. The Specify Base Name and Location dialog box appears. Enter the name and location for the generated file. The Specify Data Model dialog appears next; you can select a data model from within the current project or browse for a data model in another project.

You are then prompted for the kind of query, which is either Select, Insert, Update, or Delete. A dialog that matches the kind of query appears. These dialogs are explained in "Creating Queries" on page 239.

Using the HTML Page Wizard

The HTML Page wizard creates an empty static HTML page. The following files are generated, with filenames based on the name you specify:

Table 8-11 HTML Page Wizard Generated Files

Generated File	**File Name**
HTML file	`name.html`

To launch the HTML Page wizard, select New File(s) from the Project menu. Select HTML Page and click Add. The Specify Base Name and Location dialog box appears. Enter the name and location for the generated file. iPlanet Application Builder creates a minimal HTML page, which sets the title and colors, in the project's HTML Pages folder and allows you to start editing the file.

Using the JSP Wizard

The JSP wizard creates an empty JSP file that is used to display dynamic data-bound HTML output. The following files are generated, with filenames based on the name you specify:

Table 8-12 JSP Wizard Generated Files

Generated File	**File Name**
JavaServer page file	`name.jsp`

To launch the JSP wizard:

1. Select New File(s) from the Project menu.
2. Select JSP and click Add.

   The Specify Base Name and Location dialog box appears.
3. Enter the name and location for the generated file.

   iPlanet Application Builder creates a minimal JavaServer page, which sets the title and colors, in the project's Java Server Pages folder and allows you to start editing the file.

Using the Servlet Wizard

The Servlet wizard creates a Java code file that you can use to define your business logic. The following files are generated, with filenames based on the name you specify:

Table 8-13 Servlet Wizard Generated Files

Generated File	**File Name**
Servlet file	`name.java`

To launch the Servlet wizard:

1. Select New File(s) from the Project menu.
2. Select Servlet and click Add.

   The Specify Base Name and Location dialog box appears.
3. Enter the name and location for the generated file.

   iPlanet Application Builder creates a Java code file, which defines the package, imports, class, and the class' `defaultAction()`, `doGet()`, and `doPost()` methods, in the project's Java folder and allows you to start editing the file.
4. If you do not wish to register the JSP, uncheck the checkbox. The default setting is to register the JSP.

Using the Java File Wizard

The Java File wizard creates an empty Java code file that you can use to implement non-servlet code, such as helper or utility classes. The following files are generated, with filenames based on the name you specify:

Table 8-14 Java File Wizard Generated Files

Generated File	File Name
Java code file	`name.java`

To launch the Java Class wizard:

1. Select New File(s) from the Project menu.
2. Select Java file and click Add.

   The file is given the name `Blank`*`n`*`.java`.

   You are prompted to change the name of the file when you close it after you modify the contents.

Using the Session Bean Wizard

The session bean wizard creates the Java files and an EJB Descriptors file needed to implement a session bean. The following files are generated, with filenames based on the name you specify:

Table 8-15 Session Bean Wizard Generated Files

Generated File	File Name
EJB remote interface source file	`Ihomename.java`
EJB enterprise bean source file	`homenameBean.java`
EJB home interface source file	`IhomenameHome.java`
EJB Descriptor	`projectnameEjb.xml`
iAS specific EJB Descriptor	`ias-projectnameEjb.xml`

You place methods in the enterprise bean source file to implement application-specific behavior in an EJB. The source files generated by the EJB wizard compile but have no application-specific behavior. You will have to add code to all three source files to create useful behavior; see `http://java.sun.com:8081/ejb/docs.html` for more information.

To launch the Session Bean wizard, select New and then Session Bean from the File menu. The Specify Base Name and Location dialog box appears. Enter the string that you want to use to construct the EJB's home name for JNDI registration as well as to create names for the iAB-generated Java files. You can change the directory if you wish. By default, it is the combination of the iAS installation directory, the relative pathname of the server's application root, and one of the following:

- the EJB default package name as specified in the Java panel of the Tools menu's Development options dialog box, if a package name is specified
- the project directory if a package name is not specified in the Java panel of this dialog

The package part of the directory name, which is the part after the application root, is used as the package name for iAB-generated source files. Consider the following example, in which iAS is installed a `C:\iPlanet\ias6`, the relative pathname for the application root is `ias\Apps`, the package is `cdx`. The bean name is `CreditCheck`. The generated path and file names would be as follows:

Table 8-16 Session Bean Wizard Generated path and File Names

Generated File	File Name
EJB remote interface source file	`C:\iPlanet\ias6\ias\Apps\cdx\ICreditCheck.java`
EJB enterprise bean source file	`C:\iPlanet\ias6\ias\Apps\cdx\ICreditCheckBean.java`
EJB home interface source file	`C:\iPlanet\ias6\ias\Apps\cdx\ICreditCheckHome.java`

In this example, the JNDI name is `java:comp/env/CreditCheck`.

You can change the names of the generated files in the next dialog box. The next dialog box also allows you to specify the following bean properties and control descriptors:

Table 8-17 Bean Properties and Control Descriptors

Property or Control Descriptor	Values
Bean class name	any valid class name
Bean home interface name	any valid class name
Bean remote interface name	any valid class name
Isolation level	One of the following levels: • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE
Run mode	One of the following modes: • CLIENT_IDENTITY • SPECIFIED_IDENTITY • SYSTEM_IDENTITY
Transaction attribute	One of the following attributes: • TX_BEAN_MANAGED • TX_MANDATORY • TX_NOT_SUPPORTED • TX_REQUIRED • TX_REQUIRES_NEW • TX_SUPPORTS

The next dialog box allows you to specify whether the bean is stateless or stateful. If the bean is stateful, you can specify a timeout value for the bean, which is 10,000 seconds by default.

Using the Entity Bean Wizard

The entity bean wizard creates the Java files and EJB Descriptors needed to implement an entity bean. The following files are generated, with filenames based on the name you specify:

Table 8-18 Entity Bean Wizard Generated Files

Generated File	**File Name**
EJB remote interface source file	`Ihomename.java`
EJB enterprise bean source file	`homenameBean.java`
EJB home interface source file	`IhomenameHome.java`
EJB Descriptor	`projectnameEjb.xml`
iAS Specific EJB Descriptor	`ias-projectnameEjb.xml`

The same EJB descriptor files are shared by all EJBs.

You place methods in the enterprise bean source file to implement application-specific behavior in an EJB. The source files generated by the EJB wizard compile but have no application-specific behavior. You will have to add code to all three source files to create useful behavior; see `http://java.sun.com:8081/ejb/docs.html` for more information.

To launch the Entity Bean wizard, select New and then Entity Bean from the File menu. The Specify Base Name and Location dialog box appears. Enter the string that you want to use to construct the EJB's home name for JNDI registration as well as to create names for the iAB-generated Java files. You can change the directory if you wish. By default, it is the combination of the iAS installation directory, the relative pathname of the server's application root, and one of the following:

the EJB default package name as specified in the Java panel of the Tools menu's Development options dialog box, if a package name is specified

the project directory if a package name is not specified in the Java panel of this dialog

The package part of the directory name, which is the part after the application root, is used as the package name for iAB-generated source files. Consider the following example, in which iAS is installed a `C:\iPlanet\ias6`, the relative pathname for the application root is `ias\Apps`, the package is `cdx`. The bean name is `CreditCheck`. The generated path and file names would be as follows:

Table 8-19 Entity Bean Wizard Generated Path and File Names

Generated File	File Name
EJB remote interface source file	`C:\iPlanet\ias6\ias\Apps\cdx\ICreditCheck.java`
EJB enterprise bean source file	`C:\iPlanet\ias6\ias\Apps\cdx\ICreditCheckBean.java`
EJB home interface source file	`C:\iPlanet\ias6\ias\Apps\cdx\ICreditCheckHome.java`

In this example, the JNDI name is `java:comp/env/CreditCheck`.

You can change the names of the generated files in the next dialog box. The next dialog box also allows you to specify the following bean properties and control descriptors:

Table 8-20 Bean Properties and Control Descriptors

Property or Control Descriptor	Values
Bean class name	any valid class name
Bean home interface name	any valid class name
Bean remote interface name	any valid class name
Isolation level	One of the following levels: • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE
Run mode	One of the following modes: • CLIENT_IDENTITY • SPECIFIED_IDENTITY • SYSTEM_IDENTITY

Table 8-20 Bean Properties and Control Descriptors

Property or Control Descriptor	Values
Transaction attribute	One of the following attributes: • TX_MANDATORY • TX_REQUIRED • TX_REQUIRES_NEW • TX_SUPPORTS

Chapter 9

Compiling, Testing, and Debugging Applications

This chapter describes debugging applications as well as deploying the finished product onto one or more iPlanet Application Servers.

The following topics are described in this chapter:

- Compiling Applications
- Building EJB Stub and Skeleton Code
- Testing Applications
- Debugging Applications

Compiling Applications

You can perform operations on all the files in your application at once by choosing one of the following options from the Build menu:

Table 9-1 Build Menu Options

Build option	**Shortcut**	**Resulting action**
Build Project	F7	Compiles all files that either are new or have been changed since the last build (equivalent to `make all`)
Rebuild Project	Ctrl+Shift+F7	Removes all files generated by previous builds and then compiles everything (equivalent to `make clean all`)
Clean Project		Remove all generated files (equivalent to `make clean`)

Rebuild Project is the recommended procedure for all milestone builds, and especially before deployment.

Registering Files

iPlanet Application Builder incorporates automatic registration for project servlets.

Compiling Java files

As you add servlets, HTML pages, and JavaServer Pages (JSPs) to your application project, you can compile and run the incomplete application periodically to test the elements you have completed so far. Before you can test a servlet, you must compile it.

The Java tab of the Development Options dialog box lets you change how iPlanet Application Builder compiles your Java files. For more information, see “Setting Application Builder Options.”

Compiling a Java file does not affect a class that has already been loaded into the runtime or test servers. iPlanet Application Builder prompts you to restart your server for this reason.

The Messages window (usually located just below the Properties window) is an information window that displays the associated compilation errors, warnings, and other data about your servlets.

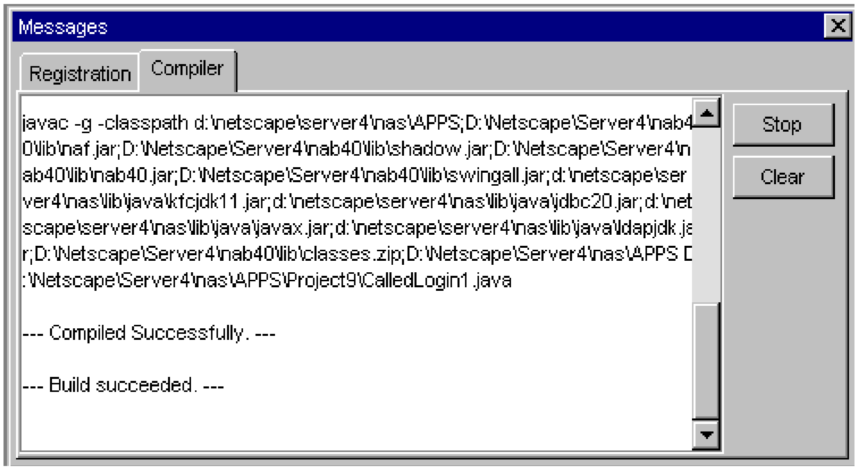
To compile a single file, perform the following steps:

1. Select the desired Java code file.
2. Choose Compile file from the Build menu.

   Messages from the compiler appear within the output message window of iPlanet Application Builder, as shown in the following illustration. If the compiler finds an error, the location of the error is shown. The messages box that iPlanet Application Builder displays allows you to double-click on a line giving the position of a compile error; iPlanet Application Builder then displays that specific line of code.

   Click Stop to stop the compile process; click Clear to reset the message window.

   If the servlet contains no syntax errors, iPlanet Application Builder displays the Compiled Successfully message.



```
javac -g -classpath d:\netscape\server4\nas\APPS;D:\Netscape\Server4\nab4
0\lib\naf.jar;D:\Netscape\Server4\nab40\lib\shadow.jar;D:\Netscape\Server4\n
ab40\lib\nab40.jar;D:\Netscape\Server4\nab40\lib\swingall.jar;d:\netscape\ser
ver4\nas\lib\java\kfcjdk11.jar;d:\netscape\server4\nas\lib\java\jdbc20.jar;d:\net
scape\server4\nas\lib\java\javax.jar;d:\netscape\server4\nas\lib\java\ldapjdk.ja
r;D:\Netscape\Server4\nab40\lib\classes.zip;D:\Netscape\Server4\nas\APPS D
:\Netscape\Server4\nas\APPS\Project9\CalledLogin1.java

--- Compiled Successfully. ---

--- Build succeeded. ---
```

To compile all your Java files, perform the following steps:

1. From the Build menu, choose Build Project.

   Messages from the compiler appear within the output message window of iPlanet Application Builder. If the compiler finds an error, the location of the error is shown. The messages box that iPlanet Application Builder displays allows you to double-click on a line giving the position of a compile error; iPlanet Application Builder then displays that specific line of code.

2. Click Stop to stop the compile process; click Clear to reset the message window.

If the servlet contains no syntax errors, iPlanet Application Builder displays the Compiled Successfully message.

NOTE You must stop and restart the webserver after you build a component

Starting and Stopping a Test Server

Your application can be run and tested without deploying to an application server. This is done by running it on a **test server**, a local version of the iPlanet Application Server on your development machine.

The test server starts or restarts automatically when you select Test Project (Ctrl+F5) from the Test menu. It also starts when you test individual files, if needed.

Alternatively, you can administer the server by hand using the following commands from the Test menu:

Table 9-2 Test Menu Commands

Build Option	Shortcut	Resulting Action
Start Server	F6	Start the test server
Stop Server	Shift+F6	Stop the test server, terminating any running processes
Restart Server	Ctrl+Shift+F6	Stop and then restart the test server, terminating any running processes

Building EJB Stub and Skeleton Code

When you select a bean and then choose Compile file(s) from the Build menu or choose Build or Rebuild Project, iPlanet Application Builder opens the EJB Stubs tab in the Messages window to display the results of the compilation. Errors during compilation appear in this window as well.

Testing Applications

You can test individual files by using a debugger, such as Visual Cafe, to set breakpoints and walk through the execution of files line by line. You can test the behavior of individual pages. Test a JSP by invoking the servlet that calls it, which also tests the runtime servlet at the same time.

Application flow, however, is the point-to-point interaction between your application and the person using it. Testing at the application level involves exercising application functionality and looking at the results as well as judging the order of the functionality with respect to other features. For example, an online market might allow a user to browse a catalog, select items, and then purchase all selected items at once at “checkout.” If your application presents a blank checkout screen before allowing your user to browse the contents of the market, your user will be confused and unsure where to go next.

This section describes the following topics:

- Testing Queries
- Testing HTML Pages and JSPs
- Testing Servlets
- Testing EJBs

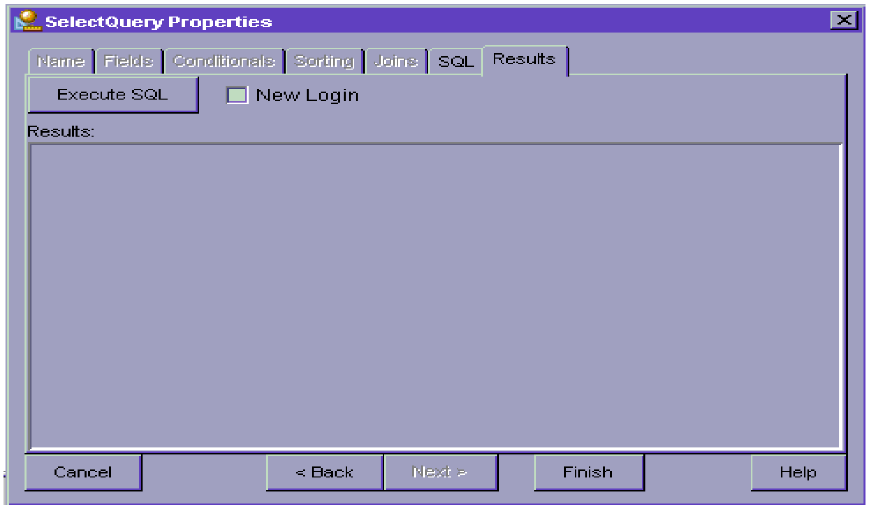
Testing Queries

iPlanet Application Builder enables you to test your queries before you deploy your application.

To test a query, perform the following steps:

1. From the File menu, choose Open to open the desired query file or double-click the query file in the project map window or within the Project window.
2. Click the Query Properties Results tab.

   iPlanet Application Builder displays the following dialog box:



3. To execute the associated SQL code for the specified query, click Execute SQL. Note that you can also right-click the query file from the Project window and select Test to perform this action or from the Test menu, select Test *MyQuery_Name*.

   If you check New Login, iPlanet Application Builder displays a dialog box that enables you to specify the following information:

   - driver
   - data source
   - user name
   - password

   If input parameters are required for this query, iPlanet Application Builder displays a dialog box for you to specify them. Note that you must place string parameters in single quotes if the query doesn't supply them.

4. Type the appropriate information for these fields and click OK.

5. To test the SQL for a new login, click New Login.

   If the SQL code contains `:<variables>`, iPlanet Application Builder displays a dialog box for you to enter the associated input parameters.

6. Click Finish if you are satisfied with the test results; otherwise, continue to edit the associated query SQL until you obtain the correct results. If iPlanet Application Builder cannot parse your SQL, all other query editing tabs are disabled except the SQL and Results tabs.

Testing HTML Pages and JSPs

At any point, you can test your page or template by viewing it in a browser.

Testing HTML Pages

Select the page you want to test, and then choose Test *Pagename* from the Test menu or type Ctrl+Shift+F5. The page appears in the default browser.

Testing JavaServer Pages

You must associate an servlet with your JavaServer Page (JSP) before it can be tested, as a JSP is meaningless without the context of data from a servlet.

Select the JSP you want to test, and then choose Test *Pagename* from the Test menu or type Ctrl+Shift+F5. If more than one servlet / EJB calls this JSP, you are prompted to choose one.

A static HTML page is generated that contains a link to the servlet / EJB that calls this JSP. This static page appears in the default browser. To test the template, click the link that calls the servlet / EJB. The results appear in the browser.

You can also test or debug a JSP by adding the Java code it produces to your project and using a debugger on that code. To generate the code:

1. Run your application and access the URL that causes the JSP to be called.

   The code is created in the following location:

   `<ias>\Apps\compiled_jsp\<appName>\<JSP_name>.java`

where `<ias>` is the location in which iAS is installed, `<appName>` is the name of your application (for example, `cdx`), and `<JSP_name>` is the name of your JSP.

2. Add this file to your project as you would any other Java file.
3. Set breakpoints or perform other debug operations on this file.

Testing Servlets

Once you've successfully designed, registered, and compiled your Java objects, you can use iPlanet Application Builder to test them. When you test a Java object, your preferred HTML browser displays an HTML page from which you can submit a request to run the object.

Before testing an object, you must compile it. For more information about how to compile your Java objects, see "Compiling Java files."

To Test a servlet, perform the following steps:

1. Open or select the servlet file.
2. Choose Test `<servlet File Name>` from the Test menu.

   iPlanet Application Builder tests the selected servlet file.

It takes a few seconds for the server to start up, even though iPlanet Application Builder immediately displays the HTML page that leads to the code being tested. Exiting iPlanet Application Builder automatically shuts down any server that was started by iPlanet Application Builder. You should wait until your Messages window displays the "Server is ready to execute Servlets" message.

Debugging Applications

If your Java object doesn’t compile correctly, or if your objects do not provide you with the expected behavior, you can use Visual Cafe to check the state your program at a particular point in its execution. For information about debugging with Visual Cafe, “Using iAB with Symantec Visual Café”.

If your Java object doesn’t compile correctly, or if your objects do not provide you with the expected behavior, you can use iPlanet Application Builder to check the state your program at a particular point in its execution via the Test and View menus or Debug toolbar.

Chapter 10

Deploying Applications

This chapter describes deploying the finished product onto one or more iPlanet Application Servers.

The following topics are described in this chapter:

- Deploying Applications
- Saving EAR Files

Deploying Applications

Deploying your application means creating a copy of it on one or more servers, usually in order to make it publicly available for use. You deploy an application from the Deployment Manager. The Deployment Manager is a tool that has been integrated into the iPlanet Application Builder. For information about the Deployment Manager, see the *Administration and Deployment Guide* for the iPlanet Application Server.

JSPs and Servlets must be registered in order to make them available to the server; this can be done at deployment time by using the JSP and Servlet Meta Editors.

Enterprise JavaBeans should be prepared for deployment by rechecking their access control lists, timeouts, and other values that need different values in a production environment. For information about how to change these values, see "Examining and Changing EJB Properties".

About Deploying Applications

You must register a machine with iPlanet Application Builder in order to deploy to it. If you employ more than one server on its own machine, you can perform load balancing and process management in order to increase application responsiveness. These issues are normally managed by the server administrator.

About File Locations

All compiled Java files, Tag libraries, JavaServer Pages (JSPs), and database access files are deployed to the application server (or servers, if there is more than one in your server configuration).

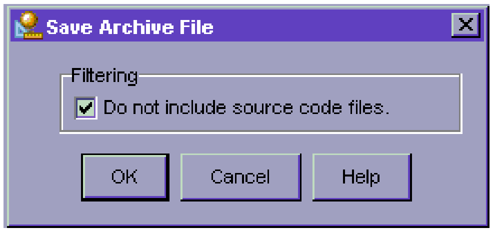
Static files that do not require interaction with the application server, such as images, static HTML pages, and JavaScript files, are deployed directly to a web server if one exists in your server configuration. These files can be served by the application server if no web server exists.

How to Deploy an Application

To deploy an application:

1. Choose Rebuild Project from the Build menu to verify that your project is up-to-date.
2. Choose Deploy from the Project menu to invoke the Deployment Manager.

   You will need to package your project as an EAR file.
3. Specify whether you want to include source files when you deploy the project.

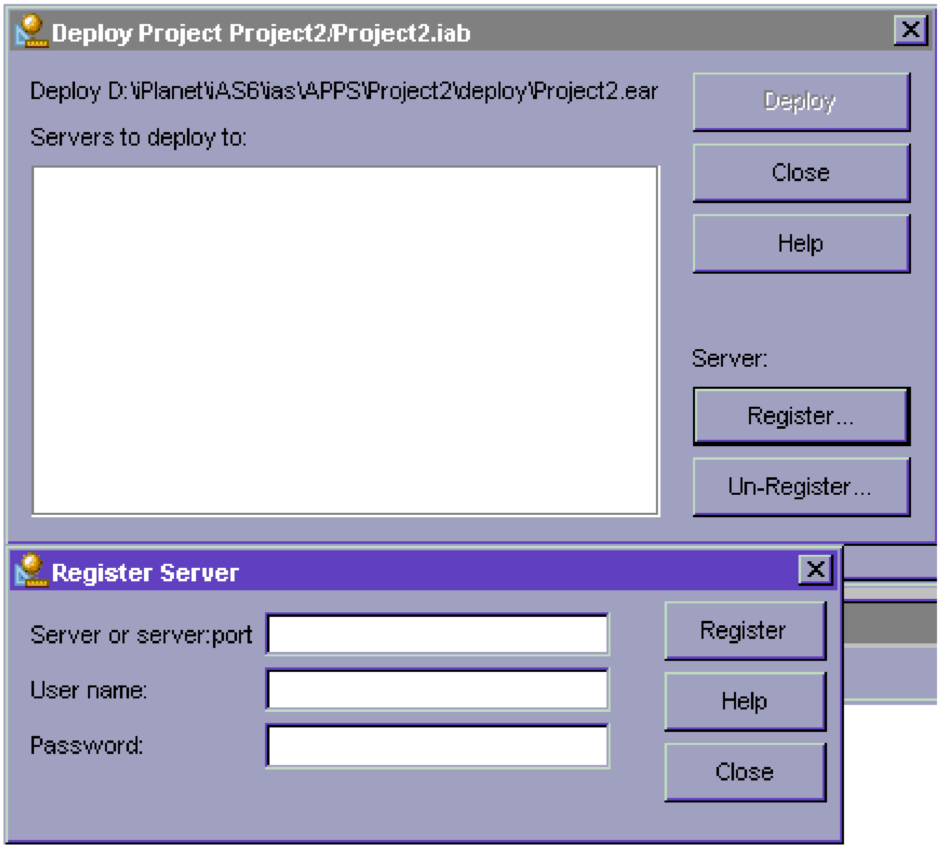

Save Archive File

Filtering

☑ Do not include source code files.

4. Register your server if it has not already been registered.

   To deploy to one or more servers, you must register the server machine with iPlanet Application Builder. If the server does not appear in the Registered Servers list, click Register. The following dialog box appears:



You must provide the following information for each deployment machine:

Table 10-1 Deployment Machine Information

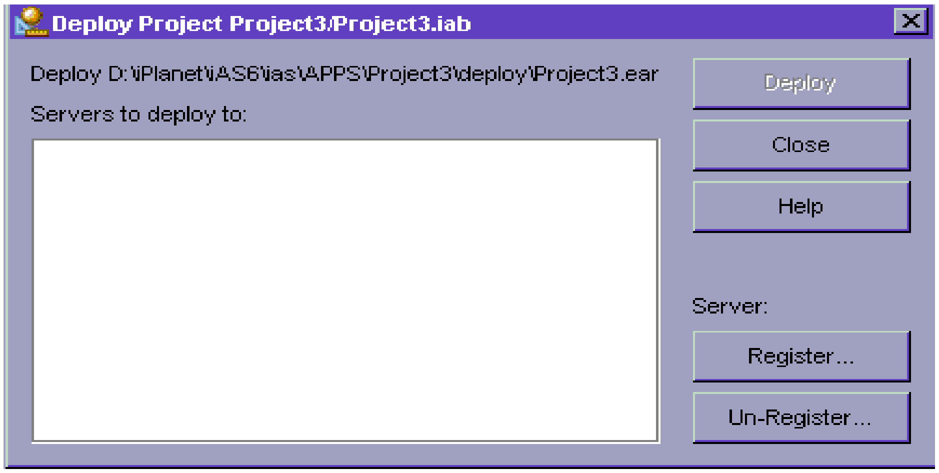
Option	Information to Provide
Server type	Specify whether this machine runs a web server in addition to or instead of an application server. If a web server exists on the machine where you are deploying your application, static entities like graphic images, static HTML pages, and JavaScript files can be served from there.
Server or server:port	Specify the machine name and, if necessary, an alternate port number, depending on how your server is configured.

Table 10-1 Deployment Machine Information

Option	Information to Provide
User name and password	Specify a valid user name and password for this machine. You can get these from the administrator who installed the software on this server. Be sure this user has permission to write to the application server and/or web server document repositories.

Enter the relevant information about each deployment machine and click Register. Click Close to dismiss this dialog box when you have registered all deployment machines.

- Select the application to be deployed, then click Deploy to deploy your application on the server.


Deploy D:\iPlanet\iAS6\ias\APPS\Project3\deploy\Project3.ear

For more detailed information about using the Deployment Manager to deploy your application, see the *Administration and Deployment Guide*.

Saving EAR Files

You can use the Package as EAR item on the Project menu to save an application as an EAR file. Specify whether you want to include source files when you deploy the application.

Select the Package Project tab from the Messages window to check that the project packaging was successful.

You can deploy the EAR file using the Deployment Manager, but not iPlanet Application Builder. You may explicitly delete this file when it is no longer needed.

NOTE An existing EAR file for an application is overwritten whenever you deploy the application.

Chapter 11

Working with Source Control

This chapter describes the source control features available within iPlanet Application Builder.

The following topics are described in this chapter:

- About Source Control
- Configuring Source Control
- Invoking Commands from the Main Menu
- Sharing Settings Among Programmers

About Source Control

Source control lets you track multiple file revisions and offers the following advantages:

- Makes compiling and building easier.

  When you build a project, you can ensure that you use the latest version (or the correct version) of all files.

- Reduces storage space.

  Changes between file versions are stored as deltas, not as complete files. This also avoids the potential confusion of keeping separate files for each version.

- Prevents editing conflicts.

  With source control, you can check out a file and gain exclusive rights to edit it. Other developers can still view the file, but only you can change it.

Before you can use source control in iPlanet Application Builder, an underlying source control utility must be installed in your development environment. You must also configure iPlanet Application Builder for the particular utility you intend to use.

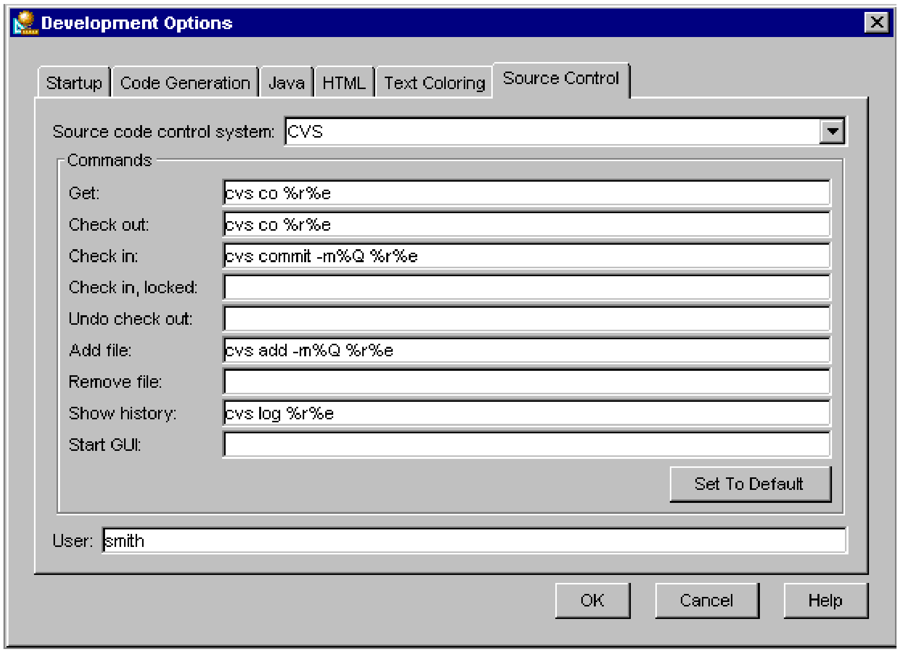
It is assumed that you a familiar with source control concepts and with the utility you intend to use. For more information, refer to the documentation that comes with your site's source control utility.

Configuring Source Control

Once a source control utility is available for you to use, you configure iPlanet Application Builder as follows:

1. Specify which source control utility to use.
2. Map the source control menu items in iPlanet Application Builder to corresponding command lines in the source control utility.

You perform these tasks from the Source Control tab in iPlanet Application Builder. From the Tools menu, choose Development Options and then click the Source Control tab. The following dialog box appears:



Source code control system: CVS

Commands

Get:	cvs co %r%e
Check out:	cvs co %r%e
Check in:	cvs commit -m%Q %r%e
Check in, locked:	
Undo check out:	
Add file:	cvs add -m%Q %r%e
Remove file:	
Show history:	cvs log %r%e
Start GUI:	

User: smith

Specifying a Source Control Utility

Often, a network administrator sets up a particular source control utility for use in a networked environment. This setup may include defining user IDs for anyone who plans to use source control. Set up may also define policies for directory names and hierarchies. Keep this in mind as you specify a source control utility.

To specify a source control utility, use the drop-down list near the top of the Source Control tab and select one of the following supported utilities:

- CVS
- PVCS
- Revision Control System (RCS), available from the Free Software Foundation
- Microsoft Visual Source Safe
- A user-defined utility

Choose the source control utility installed on your machine. If you select a user-defined source control utility, no default command-mappings are provided. In this case, you need to map the menu items from scratch.

Defining Menu Items

In the Source Control tab, the Commands area shows command-line equivalents for the items in the Source Control submenu.

For example, suppose PVCS is the selected utility, as shown in the previous figure. The Show History text field contains the command:

```
vlog %b%e
```

In this example, when you choose Show History from the Source Control submenu, iPlanet Application Builder calls the PVCS utility, which in turn executes the vlog command. The characters `%b` and `%e` are macros, which are substituted for their current values at execution time. For more information, see “Macro Definitions” on page 197.

iPlanet Application Builder supplies default command-line equivalents. However, you can edit the text fields to suit your needs. You can also restore default values by pressing the Defaults button.

Before you edit text fields on the Source Control tab, make sure you understand the following topics:

- Relationship of Text Fields to Source Control Submenu
- Default Values for Command Lines
- Macro Definitions

Relationship of Text Fields to Source Control Submenu

The following table describes the text fields on the Source Control tab and their corresponding item on the Source Control submenu:

Table 11-1 Source Control Tab Text Fields

Label in Source Control Tab	**Equivalent Itemon Submenu**	**Description**
Get	Get Latest Version ...	Updates selected files with their latest version from the source control system.
Check Out	Check Out ...	Checks out selected files from the source control system and opens them for editing.
Check In	Check In ...	Checks in selected files to the source control system, with a comment.
Check In, Locked	Check In ...	If you select the “Keep checked out” box, files remain checked out (locked) after changes are submitted.
Undo Check Out	Undo Check Out ...	Reverts the local version of selected files to the version in the source control system.
Add File	Add to Source Control ...	Adds files to the source control system.
Remove File	Remove from Source Control ...	Removes selected files from the source control system. Files are not removed from the project directory.
Show History	Show History ...	Displays revision history for selected files.
Start GUI	Source Control System ...	Displays the graphical user interface of the source control utility. This item is useful when you want to work directly with the utility.

Default Values for Command Lines

You can edit the command lines, or you can use the default values shown in the following table. A command is executed once for each of the selected files as if the command were run from the command line after making substitutions for macros. If you change a command line, the command line must conform the following conventions:

- it cannot request input; for example, it cannot prompt for a value or option
- it assumes that the command is run from the directory that contains the file

To debug your command line, you should check iPlanet Application Builder's message window to view the macro substitution. If your command does not work, you should check it outside of iPlanet Application Builder by attempting to execute the iAB-generated command from the command line.

CVS Commands

The following default commands are defined for CVS:

Table 11-2 CVS Default Commands

Label in Source Control Tab	CVS Command
Get	cvs co %r%e
Check Out	cvs co %r%e
Check In	cvs commit -m%Q %r%e
Check In, Locked	
Undo Check Out	
Add File	cvs add -m%Q %r%e
Remove File	cvs log %r%e
Show History	
Start GUI	

PVCS Commands

The following default commands are defined for PVCS:

Table 11-3 PVCS Default Commands

Label in Source Control Tab	PVCS Command
Get	get -y %b%e
Check Out	get -y -l %b%e
Check In	put -n -M%Q %b%e
Check In, Locked	put -n -M%Q -l %b%e

Table 11-3 PVCS Default Commands

Label in Source Control Tab	PVCS Command
Undo Check Out	vcs -y -u %b%e
Add File	vcs -i -n -t%Q %b%e
Remove File	del /f %b%e
Show History	vlog %b%e
Start GUI	

If you use PVCS, you must specify a configuration file, which is not included in the default commands. By default, PVCS places the `default.cfg` file in the following directory:

`<pvcs_root>\pvcs\pvcsprop\pvcs\vm`

You need to add this file to your commands using the -C option, as shown in the following examples:

Table 11-4 Example PVCS Commands with -c Option

Label in Source Control Tab	Example PVCS Command with -C option
Get	get -y -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" %b%e
Check Out	get -y -l -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" %b%e
Check In	put -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" -n -M%Q %b%e
Check In, Locked	put -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" -n -M%Q -l %b%e
Undo Check Out	vcs -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" -y -u %b%e
Add File	vcs -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" -i -t"new file" %b%e
Remove File	del /f %b%e

Table 11-4 Example PVCS Commands with -c Option

Label in Source Control Tab	Example PVCS Command with -C option
Show History	vlog -c"E:\Program Files\PVCS\pvcsprop\pvcs\vm\default.cfg" %b%e
Start GUI	

If you do not enter a comment in a PVCS dialog box, you may receive an "out of memory" error from PVCS.

Visual Source Safe Commands

The following default commands are defined for Visual Source Safe:

Table 11-5 Visual Source Safe Default Commands

Label in Source Control Tab	Visual Source Safe Command
Get	ss GET -I- %r%e
Check Out	ss CHECKOUT -I- %r%e
Check In	ss UPDATE -I- -C%Q %r%e
Check In, Locked	ss UPDATE -I- -C%Q -K %r%e
Undo Check Out	ss UNCHECKOUT -I- -G- %r%e
Add File	ss ADD -I- -C%Q %r%e
Remove File	ss DELETE -I- %r%e
Show History	ss HISTORY -I- %r%e
Start GUI	

Macro Definitions

You define source control commands using macros, which are substituted for actual values when the command is executed. You can choose from among the macros listed in the following table. Note that the examples provided are based on the following files:

- Source file is `c:\iplanet\ias6\ias\apps\cdx\ListUsers.java`
- Project file is `c:\iplanet\ias6\ias\apps\cdx\cdx.iab`

Table 11-6 Macro Definitions

Macro	Substitution For	Example
${USER}	The value of the User Name field in the Source Control tab.	
%b	Basename. The name of the working file, minus the suffix.	`c:\iplanet]ias6\ias\apps\cdx\ListUsers`
%d	Directory. The directory of the working file. It contains no filename and no drive letter. There is no trailing backslash.	`\iplanet\ias6\ias\apps\cdx\`
%e	Extension. The suffix of the working file. It begins with a dot (.), unless it is null.	`.java`
%p	Path. The directory path of the working file. Same as %d with a trailing backslash.	`\iplanet\ias6\ias\apps\cdx\`
%Q	Text from the Comment box in the Check In dialog box. This macro is case-sensitive, so use an uppercase Q.	
%r	Root. The name of the working file, minus the suffix.	`ListUsers`
%v	Volume. The volume drive of the working file. A letter and a colon.	`c:`
%x	Project path. The path of the current `.iab` file. Typically this is the project directory.	`\iplanet\ias6\ias\apps\cdx\`
%y	Project root. The name of the current `.iab` file.	`cdx`
%%	Percent sign. Use this for a literal % in the command.	

Setting Up the CVS Environment

To use CVS commands from iPlanet Application Builder, you must map your application source directory to a CVS archive. Typically, you manually copy the CVS directories and their contents from the client source tree to your application directory.

For example, to set up CVS source control for your Java code files, you would perform the following steps:

1. Set your CVSROOT environment variable.

   ```
   set CVSROOT=:pserver:user@cvsserver:/m/src
   ```

2. Change to a directory in which CVS is already set up and add the application directory tree at each level.

   ```
   cd D:\cvssource
   mkdir iplanet
   cvs add iplanet
   cd iplanet
   mkdir ias6
   cvs add ias6
   cd ias6
   mkdir ias
   cvs add ias
   cd ias
   mkdir apps
   cvs add apps
   cd apps
   mkdir cdx
   cvs add cdx
   ```

   If you use a FAT partitioned disk, you cannot use case-sensitive names because CVS is case sensitive but FAT directory names are not. In this case, you should use an NTFS partition.

3. From the command line, add a single file into your application directory.

   ```
   cd cdx
   cvs add dummy.txt
   ```

4. From the top-level, commit the file, which causes CVS to create a repository tree.

   ```
   cd D:\cvssource
   cvs commit
   ```

5. From your CVS client tree, such as `D:\cvssource`, copy the applications directory, such as `iPlanet` (and includes the file you just added), to your server directory; for example, `D:\iPlanet`.

Setting Up Visual Source Safe

To create a Visual Source Safe project and add a iPlanet Application Builder project, perform the following tasks:

1. Launch the Visual SourceSafe Explorer.

2. Right-click on the root folder (`$/`) and select “Create Project.”
3. Specify the name of your Visual SourceSafe project; for example, `NABApps`, and click OK.
4. Right click on “Set Working Folder.”
5. Specify or navigate to the directory that contains your iPlanet Application Builder projects; for example, `D:\iplanet\ias6\ias\apps`.
6. For each iPlanet Application Builder project you want to place under source control, right-click on your Visual SourceSafe project, select “Create Project” and specify the name of the iPlanet Application Builder project.

   For example, if you specified `D:\iplanet\ias6\ias\apps` in the previous step, when you specify `cdx`, `D:\iplanet\ias6\ias\apps\cdx` becomes the working folder.

You can add users for Visual SourceSafe using the Visual SourceSafe Admin program. The program creates an ss.ini file in the `Users` subdirectory in which Visual SourceSafe is installed. The file should contain at least the following entries:

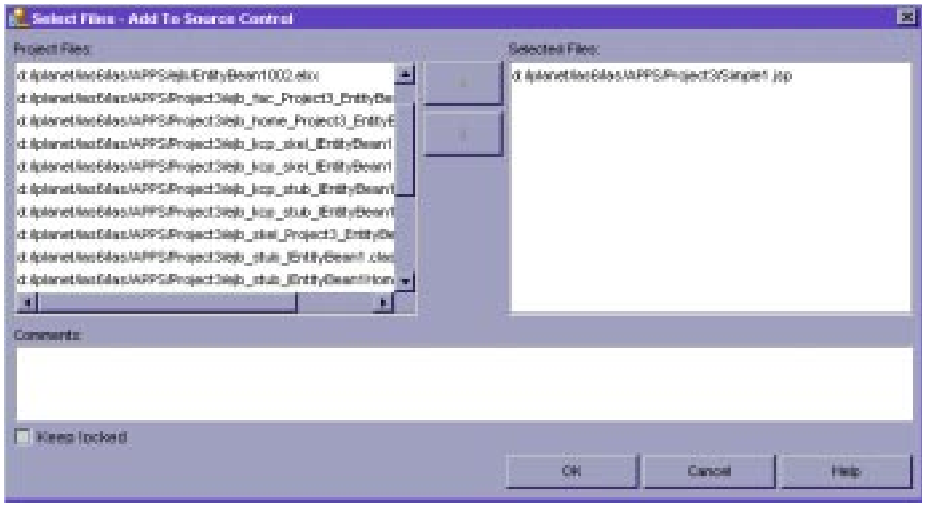
```
; Your current SourceSafe project
Project = $iabapps
Force_Prj = Yes
...
[$/iabapps]
Dir (your_host_machine)=D:\iplanet\ias6\ias\apps
```

If you are creating multiple accounts, you can simply replace each user’s `ss.ini` file with the one you created initially by copying it to their `Users` subdirectory.

Invoking Commands from the Main Menu

You invoke source control commands by pulling down the Tools menu, choosing the Source Control submenu, and then choosing one of its menu options. You can select one or more files from the Project window first, but selecting files is optional.

When you invoke a source control command, a Select Files dialog box appears. For example, suppose you are working in the HelloWorld project. If you highlight `index.html` in the Project window and then choose Add to Source Control, the following dialog box appears:



```
d:\iplanet\ias6\ias\APPS\ejb\EntityBean1002.ebx
d:\iplanet\ias6\ias\APPS\Project3\ejb_fac_Project3_EntityBe
d:\iplanet\ias6\ias\APPS\Project3\ejb_home_Project3_EntityE
d:\iplanet\ias6\ias\APPS\Project3\ejb_kcp_skel_IEntityBean1
d:\iplanet\ias6\ias\APPS\Project3\ejb_kcp_skel_IEntityBean1
d:\iplanet\ias6\ias\APPS\Project3\ejb_kcp_stub_IEntityBean1
d:\iplanet\ias6\ias\APPS\Project3\ejb_kcp_stub_IEntityBean1
d:\iplanet\ias6\ias\APPS\Project3\ejb_skel_Project3_EntityBe
d:\iplanet\ias6\ias\APPS\Project3\ejb_stub_IEntityBean1.clas
d:\iplanet\ias6\ias\APPS\Project3\ejb_stub_IEntityBean1Hom
```

```
d:\iplanet\ias6\ias\APPS\Project3\Sample1.jsp
```

The Selected Files window, on the right, lists the full path names of any files you had previously selected. The window on the left lists all other files in the project.

Using the Select Files dialog box, you can change the set of files that you want to affect. For example, if you want to add more files to the source control system, you can highlight other listed files and then click the right arrow button. When you press OK, you apply the source control command to whatever files are listed in the Selected Files window.

Sharing Settings Among Programmers

In many cases, several programmers work on the same project, but each programmer might be using a different source control configuration. It may be desirable for everyone to share the same configuration settings.

To share these settings, one programmer can export them from the registry to a file. Other programmers can then import the resulting file into their respective registries.

The following procedure describes how to export source control settings:

1. Open the iPlanet registry editor by typing `regedit` from a DOS command prompt.

   The registry editor appears, and it displays the keys and values that apply to the iPlanet Application Server.

2. Locate the key denoted by the path

   `HKEY_LOCAL_MACHINE\SOFTWARE\iPlanet\Application Builder\6.0\SCCS`

   This key contains the command-line settings for source control in iPlanet Application Builder.

3. Select the SCCS folder.

4. From the Registry menu, choose Export Registry File.

5. Specify the name and location of the file.

   Do not specify a suffix. The `.reg` suffix is automatically appended to the file name.

Once the source control settings are exported, you can import them in either of the following ways:

- Double-click the previously exported `.reg` file.
- From a command prompt, run the command `regedit filename`, where *filename* is the previously exported `.reg` file.

Appendix A

Component Reference

This appendix describes all components available on the palette.

About Components

Components are reusable objects that you can place on an HTML page or JavaServer Page to perform a certain task. The behavior and appearance of components are determined by their properties.

Components are Java objects called **Editor Beans** that generate the required HTML and JavaScript code, depending on the environment into which they are dropped. For example, if you drop a form component like PushButon onto a blank page, a new form is automatically created, though the same component uses an existing form if dropped onto one. Note that some of the code generated by Editor Beans should not be edited; see “Using Source View”.

Data-Bound Properties

Some components have properties which allow their behavior and appearance to be controlled at run-time by a servlet or JSP tag (rdbm:field). These are called **data-bound properties**. They only appear on components that have been inserted into a JSP, as they serve no purpose on a static HTML page.

The following components contain data-bound properties that can be configured to represent data returned from a servlet or JSP tag for display:

Table A-1 Data Bound Properties

HTML tab	Data-Bound Image Link, Data-Bound Link, List

Table A-1 Data Bound Properties *(Continued)*

Form tab	CheckBox, Drop-Down List, List Box, Radio Button Group, TextField, Text Area
RDBM tab	RDBM Taglib Directive, Query, Loop, Field, Param, Close, Execute, GoRecord

Target Window

Some components provide hypertext links to load other URLs or called servlets. These components can display the resulting page in the current window or in another window or frame by setting the Target property.

The following values are valid for the Target property:

Table A-2 Target Property Valid Values

`_self`	Display in the same frame or window.
`_top`	Display in the full body of the window, overriding any existing frames.
`_blank`	Display in a new blank window.
`_parent`	Display in the same frameset as the current link, removing any child frames.
`none`	Display in current window. Same as `_self`.
`custom`	Custom target.

Consult an HTML reference for more details.

Palette Tabs

Components appear by default on the following palette tabs:

HTML tab.

Data-Bound Link	Hypertext link
Image	HTML image
Data-Bound Image Link	HTML image that serves as a hypertext link
DropDown Link	Hyperlinked combo-box navigation widget
Link Target	Named hypertext target
Horizontal Line	Horizontal rule (`<hr>`)
Table	HTML table
Loop	Data-bound repeating container
List	Data-bound list

Form tab.

Form	HTML form container
Push Button	HTML form Submit or Reset button
Text Field	HTML form text field
Text Area	HTML form text field, multiple lines
List Box	HTML form list box
Drop-Down List	HTML form drop-down selector list
Radio Button Group	HTML form radio-button
CheckBox	HTML form checkbox
Custom Script	Custom script

JSP tab.

JSP Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page.
JSP Include Directive	Includes a file or text code when the JSP page is translated.
JSP Page Directive	Defines attributes that apply to an entire JSP page.
JSP Expression	Contains an expression valid in the scripting language used in the JSP page.
JSP Scriptlet	Contains a code fragment valid in the page scripting language.
JSP Declaration	Declares a variable or method valid in the scripting language used in the JSP page.
JSP Include	Sends a request to an object and includes the result in a JSP file.
JSP Forward	Forwards a client request to an HTML file, JSP file, or servlet for processing.
JSP UseBean	Locates or instantiates a Bean with a specific name and scope.
JSP SetProperty	Sets a property value or values in a Bean.
JSP GetProperty	Gets a property value or values in a Bean.
JSP Param	Provides key/value information tothe JSP Include, JSP Forward, and JSP Plugin tags.
JSP ParamList	Provides a grouping of JSP Param tags.
JSP Plugin	Downloads a Java plugin to the client web browser to execute an applet or Bean.
JSP Fallback	Indicates the content to be used by the client browser if the plugin cannot be started.

RDBM tab.

Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page.
Query	Declares a use of a ResultSet.
Loop	Loops through the contents of a ResultSet.
Field	Displays a particular column in the ResultSet.
Param	Sets a parameter on a RowSet

Close	Releses resources back to the system.
Execute	Executes the identified query.
GoRecord	Optionally executes the specified query and moves the ResultSet to the indicated record.

Components

The following sections describe each component in the iPlanet Application Builder JSP Palette tab.

CheckBox

The CheckBox component provides a form field in the form of a checkbox. If the box contains a check when the form is submitted, the field's value (see the Value property below) is submitted with the form. Application users can check or uncheck the box by clicking on it at runtime.

This component can optionally be bound to output data from a servlet. For more information, see “Binding JavaServer Page Components to Dynamic Data”.

This component resides on the Form tab by default.

Table A-3 Form Tab Default Components

Property	Values	Description
(Name)	String	Name of this component.
Checked	`true` or `false`	Whether the box is checked by default.
DataField	Field in ResultSet	Name of the data set field submitted with the form. If the box is checked, the argument `DataField=”value”` is submitted with the form (see the Value property below).
ResultSet	Row set	Name of the servlet ResultSet used to populate this component. Adding data to this field causes the `DataField` and `Display Format` properties to appear.
Display Format	Format	Optional format mask for dynamic data.

Table A-3 Form Tab Default Components

Property	Values	Description
Value	String	Value of field, submitted with form if box is checked. This field is entered as `VALUE="value"` in the HTML code representation of this component.

Custom Script

The Custom Script component provides a framework for you to develop your own custom scripts in JavaScript. Drop this component onto your page and then edit it by clicking the Source tab.

Data-Bound Image Link

The ImageLink component is similar to an Image component, except that it also functions as a hypertext link to another page or to a servlet.

This component can optionally be bound to the results of a query.

Table A-4 Image Link Components

Property	Values	Description
(Name)	String	Name of this component.
Alignment	`ABSBOTTOM`, `ABSMIDDLE`, `BOTTOM`, `LEFT`, `MIDDLE`, `RIGHT`, `TEXTTOP`, `BASELINE`, or `TOP`	Image alignment with respect to surrounding text. For details, consult an HTML reference.
Alternate Text	String	Text which appears instead of the specified image on browsers that do not support images. This string often appears as bubble help when the cursor hovers over the image. Not used when data-bound.
Alternate Text DataField	Field in ResultSet	Name of the field in the data set to display as alternate image text.
Border	Integer	Width of a border surrounding the image, in pixels.
ResultSet	Row set	Name of the result set used to populate this component. Adding data to this field causes the `DataField` properties to appear.

Table A-4 Image Link Components *(Continued)*

Property	Values	Description
Destination URL	URL	Destination of this hyperlink. This URL is loaded when the user clicks the image.
Height	Integer	Height of the image, in pixels. Specifying this property allows your page to load faster.
Horizontal Margin	Integer	Indentation from the left of the containing page, frame, or table cell to the left side of the image, in pixels.
Image Source	URL	Location of the source file for this image. Not used when data-bound.
Image Source DataField	Field in ResultSet	Name of the field in the ResultSet to display as the location of the source file for this image.
Target	Target Window	Name of the window or frame where the results of this hyperlink should appear.
Vertical Margin	Integer	Indentation from the top of the containing page, frame, or table cell to the top of the image, in pixels.
Width	Integer	Width of the image, in pixels. Specifying this property allows your page to load faster.

Data-Bound Link

The Link component provides a hypertext link to another page or to a servlet.

This component can optionally be bound to the results of a query. For more information, see “Binding JavaServer Page Components to Dynamic Data”.

Table A-5 Data Bound Output Data

Property	Values	Description
DataField	Field in row set	Name of the field in the data set to display as link text.
ResultSet	Row set	Name of the result set used to populate this component. Adding data to this field causes the `DataField` and `Display Format` properties to appear.

Table A-5 Data Bound Output Data

Property	Values	Description
Destination URL	URL	Destination of this hyperlink. This URL is loaded when the user clicks the link.
Display Format	Format	Optional format mask for dynamic data.
Link Text	String	Text that appears on the screen. This text forms the basis of the hyperlink, meaning that it lies inside the <a> container. This property is not used when the component is data-bound.
Target	Target Window	Name of the window or frame where the results of this hyperlink should appear.

DropDownLink

The DropDownLink component defines a set of possible hypertext links in a drop-down selectable list. When an item is selected, the browser then loads the page for that item.

Table A-6 Drop Down Link Properties

Property	Values	Description
(Name)	String	Name of this component.
Items	Component-defined list entries	A list of entries for the drop-down list, including the label to display and the destination URL where the label points. You must use the custom property editor.

Drop-Down List

The Drop-Down List component provides a list of values, of which one can be selected at run-time from a drop-down list box.

This component can optionally be bound to the results of a query.

Note that you must set two data bindings, including a set of items (Items DataSet) and the item selected initially (Selection ResultSet). To set multiple data fields per option, use Labels Expression and Selection Criteria.

Table A-7 Drop Down List Properties

Property	Values	Description
(Name)	String	Name of this component.
Items	Component-defined list entries	A list of entries for the list. For best results, use the custom property editor. This property is not used when the component displays dynamic data (see the `Items ResultSet` property).
Items ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field enables all of the data binding Labels DataField, Labels Display Format, Expression, Maximum Items, Selection DataField, and Selection ResultSet properties.
Labels DataField	Depends on `Items ResultSet`	Name of the field in the Items `ResultSet` to display in the list. Corresponds to `Values DataField`, which is the value submitted with the form.
Labels Display Format	Format	Optional format mask for dynamic data.
Labels Expression	expression	Data-bound multiple-value expression for labels. The properties `Labels DataField` and `Labels Display Format` are only used if this property is blank. For best results, use the custom property editor.
Maximum Items	Integer	The maximum number of items to be displayed.
Multiple	`True` or `false`	`True` if the user can select multiple entries; `false` if only one entry selection is allowed.
Selection Criteria	Java expression	Expression that identifies the item selected initially. The `Selection ResultSet` and `Selection DataField` properties are only used if this property is blank.
Selection DataField	Field in row set	Name of the field in the `ResultSet` that identifies the item that was initially selected.
Selection ResultSet	Row set	Name of the servlet result set used to identify the item that was initially selected.
Size	Integer	Number of items visible at a time.
Values DataField	Depends on `Items ResultSet`	Name of the field in the `Items ResultSet` to match as a value to the associated `Labels DataField` and submitted with the form.

Field

The Field component defines an element for displaying dynamic text. For example, this component could reference a query which looks up a user's first name based on a provided ID and password; the resulting name is then provided to the page and displayed on the screen.

This component must be bound to output data from a useQuery.

Table A-8 Field Properties

Property	Values	Description
DataField	Field in row set	Name of the field in the row set to display as normal page text.
ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field causes the `DataField` and `Display Format` properties to appear.
Display Format	Format	Optional format mask for dynamic data.

Form

The Form component is the container for HTML form elements. This component holds components for data input, as well as other page elements as necessary.

Note that the Form component itself displays nothing on the page at runtime. You must add other components to a form, including a way to submit the form, such as the PushButton component.

Table A-9 Form Properties

Property	Values	Description
(Name)	String	Name of this component.
Action	URL	Action to perform when the form is submitted, e.g. when the Submit button is clicked. The action is specified in the form of a web address or servlet. For best results, use the custom property editor.

Table A-9 Form Properties *(Continued)*

Property	Values	Description
Encoding	`application/x-www-form-urlencoded` or `multipart/form-data`	MIME encoding for this component.
Method	`Get` or `Post`	Form submission method. For more information, consult an HTML reference.
Target	Target Window	Name of the window or frame where the results of the form submission appear.

Horizontal Line

The Horizontal Line component places a horizontal rule (`<hr>`) on your page.

Table A-10 Horizontal Line Properties

Property	Values	Description
Alignment	`CENTER`, `LEFT`, or `RIGHT`	Align to the center, left, or right side of the containing page, frame, table cell, or list item.
No 3-d Shading	`true` or `false`	If `false`, a dropped shade is applied to the line.
Thickness	Integer	Vertical thickness of the line, in pixels.
Width	Integer or Percentage	Width of the line expressed as either a number of pixels or a percentage of the containing page, frame, table cell, or list item. A value of 100% fills the container horizontally.

Image

The Image component defines a graphic image in GIF or JPG format. Note that this component is not data-bound. For a dynamic version of this component, use Data-Bound Image Link.

Table A-11 Image Properties

Property	Values	Description
(Name)	String	Name of this component.
Alignment	`ABSBOTTOM`, `ABSMIDDLE`, `BOTTOM`, `LEFT`, `MIDDLE`, `RIGHT`, `TEXTTOP`, or `TOP`	Image alignment with respect to surrounding text. For details, consult an HTML reference.
Alternate Text	String	Text which appears instead of the specified image on browsers that do not support images. On browsers that do support images, this string often appears as bubble help when the cursor hovers over the image.
Border Width	Integer	Width of a border surrounding the image, in pixels.
Height	Integer	Height of the image, in pixels, or the percentage of page height. Specifying this property allows your page to load faster.
Horizontal Margin	Integer	Indentation from the left of the containing page, frame, or table cell to the left side of the image, in pixels.
Is Map	`True` or `false`	Specify whether this image is a clickable image map. For more information, consult an HTML reference.
Low Res. Source	URL	Source for a low-resolution (often black and white) version of the image. This version loads and displays first, allowing the user to make decisions and continue without waiting for images to download.
Map Name	String	Indicates the name of a server-side image map. For more information, consult an HTML reference.
Source	URL	Location of the source file for this image.
Vertical Margin	Integer	Indentation from the top of the containing page, frame, or table cell to the top of the image, in pixels.
Width	Integer	Width of the image, in pixels, or the percentage of page width. Specifying this property allows your page to load faster.

These events do not appear on the Events tab in the property inspector. To handle an event, edit the component in Source view and add an appropriate argument to the `<img>` tag. The following example instructs the image to handle the onLoad event by calling `imageLoaded()`:

```
<IMG SRC="image.gif" ONLOAD="imageLoaded()">
```

Link Target

The Link Target component defines a named mid-page URL target, specified with the HTML code `<a name="name">`. You can link to this target with a # separator, with a URL of the form `pageURL#name`.

Table A-12 Link Target Properties

Property	Values	Description
(Name)	String	Name of this target, used as `name` above.

List

The List component defines an HTML ordered list `<ol>` or unordered list `<ul>`.

This component is intended to be bound to output data from a query. For more information, see “Binding JavaServer Page Components to Dynamic Data”.

Table A-13

Property	Values	Description
DataField	Field in row set	Name of the field in the row set to display as list items.
ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field causes the `DataField` and `Display Format` properties to appear.
Display Format	Format	Optional format mask for dynamic data.

Table A-13

Property	Values	Description
List Type	Itemized	Type of list. List types include: Bulleted - circles (filled bullets) Bulleted - discs (empty bullets) Bulleted - squares Numbered - Arabic (1, 2, 3) Numbered - Roman capitals (I, II, III) Numbered - Roman lowercase (i, ii, iii) Lettered - capitals (A, B, C) Lettered - lowercase (a, b, c)
Maximum Rows	Integer	The maximum number of rows of data to be displayed.

List Box

The List Box component provides a list of values in a form selector box, of which one or more can be selected at run-time.

This component can optionally be bound to the results of a query. For more information, see “Binding JavaServer Page Components to Dynamic Data”.

Note that you must set two data bindings, including a set of items (Items ResultSet) and the item selected initially (Selection ResultSet). To set multiple row fields per option, use Labels Expression and Selection Criteria.

Table A-14 List Box Properties

Property	Values	Description
(Name)	String	Name of this component.
Items	Component-defin ed list entries	A list of entries for the list. For best results, use the custom property editor. This property is not used when the component displays dynamic data (see the `Items ResultSet` property).
Items ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field enables the Labels DataField, Labels Display Format, Labels Expression, Maximum Items, Selection DataField, Selection ResultSet, and Values DataField properties.
Labels DataField	Depends on Items ResultSet	Name of the field in the `Items` ResultSet to display in the list. Corresponds to `Values DataField`, which is the value submitted with the form.

Table A-14 List Box Properties *(Continued)*

Property	Values	Description
Labels Display Format	Format	Optional format mask for dynamic data.
Labels Expression	expression	Data-bound multiple-value expression for labels. The properties `Labels DataField` and Labels `Display Format` are only used if this property is blank. For best results, use the custom property editor.
Maximum Items	Integer	The maximum number of items to be displayed.
Multiple	True or false	`True` if the user can select multiple entries; `false` if only one entry selection is allowed.
Selection Criteria	Java expression	Expression that identifies the initially selected item(s). The `Selection ResultSet` and `Selection DataField` properties are used if this property is blank.
Selection DataField	Field in row set	Name of the field in the row set to display.
Selection ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field displays the `DataField` and `Display Format` properties.
Size	Integer	Number of items visible at a time.
Values DataField	Depends on Items ResultSet	Name of the field in the `Items ResultSet` to match as a value to the associated `Labels DataField` and submitted with the form.

Loop

The Loop component represents a repeating row of data from a query. The component replicates itself for each row of data to be displayed. For example, if a Loop component defines a row in an HTML table, at runtime the table contains one row for each row of data.

Note that the Loop component itself displays nothing on the page; it only repeats according to the rules specified by its properties. Insert other components into the tile to display one or more fields in each row.

Table A-15 Loop Properties

Property	Values	Description
Query Name	Row set	Name of the ResultSet used to populate this component.
Scope	page, request, session, or application	Specifies the scope of visibility of the loop
StartRecord	integer	First record to display the result set.
Loop ID	String	Name of the loop with in the scope.

PushButton

The PushButton component defines a button for an HTML form.

Table A-16 Push Button Properties

Property	Values	Description
(Name)	String	Name of button (submitted with the form)
Button Type	`Submit`, `Reset`, `Button`, or `Image`	Type of button. `Submit` button sends the form to the server for processing. `Reset` button resets all fields to their default values. `Button` causes the button to perform only the actions specified with the `onClick` event (useful for creating buttons that do something other than submitting a form). `Image` is a button that displays a graphic instead of text on a button.
Label	String	Label that appears on the button. Submitted with the form if the Name property is specified.
Image Source	String	URL of the graphic file to display. Only active if the Button type is image.

Table A-16 Push Button Properties *(Continued)*

Property	Values	Description
Image Alignment	`LEFT`, `RIGHT`, `TOP`, `ABSMIDDLE`, `ABSBOTTOM`, `TEXTTOP`, `MIDDLE`, `BASELINE`, or BOTTOM	The kind of alignment you want to use for a graphic..

Radio Button Group

The Radio Button Group component defines a radio-style selector switch with two or more entries, for use in a form. Only one selection is allowed.

This component can optionally be bound to the results of a query. For more information, see “Binding JavaServer Page Components to Dynamic Data”.

Note that you may set two data bindings, including a set of items (`Items ResultSet`) and the item selected initially (`Selection ResultSet`). To set multiple data fields per option, use `Labels Expression` and `Selection Criteria`.

Table A-17 Radio Button Group Properties

Property	Values	Description
(Name)	String	Name of this component.
Labels DataField	Depends on Items ResultSet	Name of the field in the Items `ResultSet` to display as labels for the radio buttons, matched with `Values DataField` as the value submitted with the form.
Labels Display Format	Format	Optional format mask for dynamic data.

Table A-17 Radio Button Group Properties *(Continued)*

Property	Values	Description
Labels Expression	expression	Data-bound multiple-value expression for labels. The properties `Labels DataField` and Labels `Display Format` are only used if this property is blank. For best results, use the custom property editor.
Maximum Items	Integer	The maximum number of items to be displayed.
Items	Component-defined list entries	A list of entries for the radio button group. For best results, use the custom property editor. This property is not used when the component displays dynamic data (see the `Items ResultSet` property).
Items ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field causes the Labels DataField, Labels Display Format, Labels Expression, Maximum Items, Selection DataField, Selection ResultSet, and Values DataField properties to appear.
Selection Criteria	Java expression	Expression that identifies the radio button selected initially. The `Selection ResultSet` and `Selection DataField` properties are used if this property is blank.
Selection DataField	Field in ResultSet	Name of the field in the row set used to determine which radio button is selected by default.
Selection ResultSet	Row set	Name of the ResultSet used to populate this component. Choosing a data set in this field causes the `DataField` and `Display Format` properties to appear.
Values DataField	Depends on Items ResultSet	Name of the field in the `Items ResultSet` to match as a value to the associated Labels `DataField` and submitted with the form.

Table

The Table component places a table on the page.

While the Table component itself is not data-bound, its contents can be bound to output data from a servlet. Each row of a table is defined by a TR (table row) tag. Each cell is defined by a TD (table data) tag. Right click on a cell to insert or delete rows or cells. For more information, see “Binding JavaServer Page Components to Dynamic Data”. In particular, you can use the Loop component to populate rows of the table dynamically and the Field component to populate individual cells (such as header cells).

Table A-18 Table Properties

Property	Values	Description
Align	`left, right` or `center`	Alignment of table in relation to the containing page, frame, or table cell.
Background Color	Color	Table background color.
Border	Integer	Width of the border surrounding each cell, in pixels. Set to 0 for no border.
Cell Padding	Integer	Distance between cell border and contents, in pixels.
Cell Spacing	Integer	Distance between individual cells, in pixels.
Columns	Integer	Number of columns in the table, used to speed page layout.
Height	Integer or Percentage	Minimum table height, in pixels or percentage of visible screen.
Horizontal Margin	Integer	Horizontal margin between table and surrounding text, in pixels.
Vertical Margin	Integer	Vertical space between table and surrounding text, in pixels.
Width	Integer or Percentage	Table width, in pixels or expressed as a percentage of the width of the containing page, window, or cell.

Text Area

The Text Area component displays text and enables multiple-line text entry in a form.

This component can optionally be bound to the results of a query. For more information, see “Binding JavaServer Page Components to Dynamic Data”.

This component can optionally be validated when a value is entered. For more information, see “Validating Data Entered on a Form” in “Creating Presentation Layout.”

Table A-19 Text Area Properties

Property	Values	Description
(Name)	String	Name of this component.

Table A-19 Text Area Properties

Property	Values	Description
DataField	Field in row set	Name of the field in the row set to display as the value of the text area initially.
ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field causes the `DataField` and `Display Format` properties to appear.
Display Format	Format	Optional format mask for dynamic data.
Height	Integer	Height of visible text area, in lines.
Validation	Itemized	Type of validation to perform, or `(none)` to disable this feature. If `(Custom)`, use the `Validation Expression` property to perform validation.
Validation Expression	JavaScript code	Custom validation expression. Use if the `Validation property` is set to `(Custom)`.
Value	String	Initial default value of the component.
Value Required	`true` or `false`	If `true`, form submission is rejected until a value is entered. Combine with `Validation` to ensure proper values.
Width	Integer	Width of text area, in characters.
Wrapping	`OFF`, `HARD`, or `SOFT`	Method for handling input lines longer than the width of the component. `HARD` inserts a newline at the end of each line, `SOFT` indicates that the value is to be treated as one line, and `OFF` disables text wrapping altogether.

TextField

The TextField component displays text and enables a single line of text entry on a form.

This component can optionally be bound to the results of a query. For more information, see“Binding JavaServer Page Components to Dynamic Data”.

This component can optionally be validated when a value is entered. For more information, see “Validating Data Entered on a Form” in “Creating Presentation Layout.”

Table A-20 Text Field Properties

Property	Values	Description
(Name)	String	Name of this component.
DataField	Field in row set	Name of the field in the row set to display as the value of the text field initially.
ResultSet	Row set	Name of the ResultSet used to populate this component. Adding data to this field causes the `DataField` and `Display Format` properties to appear.
Display Format	Format	Optional format mask for dynamic data.
Length	Integer	Width of text area, in characters.
Maximum Length	Integer	Number of characters allowed in the input. If greater than Length, the cursor scrolls horizontally while text is being entered that exceeds Length.
Type of field	`TEXT`, `PASSWORD`, or `HIDDEN`	Nature of the text in this field. `TEXT` indicates plain text, and `PASSWORD` indicates that the value is obscured (characters appear as asterisks). `HIDDEN` indicates that the field is not visible at runtime in the browser.
Validation	Itemized	Type of validation to perform, or `(none)` to disable this feature. If `(Custom)`, use the Validation Expression property to perform validation.
Validation Expression	JavaScript code	Custom validation expression. Use if the Validation property is set to `(Custom)`.
Value	String	Initial default value of the component.
Value Required	`True` or `false`	If `true`, form submission is rejected until a value is entered. Combine with Validation to ensure proper values.

Appendix B

Using the Application Builder with Third-Party Tools

This appendix describes the interoperability considerations for using iPlanet Application Builder with third party tools.

The following topics are discussed in this appendix:

- Using iAB with Symantec Visual Café
- Using iAB with Macromedia Dreamweaver™
- Using iAB with the Visual Cafe Java Editor

Using iAB with Symantec Visual Café

You can use Symantec Visual Cafe™ to edit, compile, and debug your Java servlets. After you create a iPlanet Application Builder project, you can perform the following steps to debug it with Visual Cafe:

1. Create a Visual Cafe Project.
2. Create a dummy class to invoke the Application Server Java engine.
3. Set up the project configuration:
4. Invoke iAS with your Visual Cafe project.
5. Use Visual Cafe to debug your application.

The following sections describe these steps in more detail and show some common Visual Cafe Debugging Techniques.

If iAB complains that the DLL versions do not match when you use Visual Cafe, you must copy the `javai.dll` and `javai_g.dll` files from the `<VisualCafe>\java\bin` directory to the `<iab60>\bin` directory where `<VisualCafe>` is the directory in which Visual Cafe has been installed and `<iAB60>` is the directory in which iAB has been installed.

Creating a Visual Cafe Project

1. Open Visual Cafe.
2. Choose the File – New Project... menu item.
3. In the New Project dialog, select the "Empty Project" icon and press the OK button.

Creating a StartEngine Class

1. Choose the Insert – Class... menu item.
2. Specify the class name "StartEngine" in the name field and choose the Finish button.
3. Open the `StartEngine.java` file and add the following method to the `StartEngine` class:

```
 public static void main (String args[])
{
com.kivasoft.engine.Engine.main (args);
}
```

Configuring the Project

1. Choose the Project – Options... menu item.
2. Select the Project tab of the Options dialog box, and perform the following steps:
    - Select "Application - A program that requires java.exe to run" as your Project Type.
    - Specify "StartEngine" as your Main Class.
    - Specify "`-d`" for Program arguments.

3. Select the Directories tab of the Options dialog box, select "input class files" from the drop down list and add the following directories and jar files.

   The following lines assume you have installed iAS in the c:\iPlanet\Server4 directory:

   ```
   c:\iPlanet\ias6\ias\APPS
   c:\iPlanet\ias6\ias\lib\java\kfcjdk11.jar
   c:\iPlanet\ias6\ias\lib\java\jdbc20.jar
   c:\iPlanet\ias6\ias\lib\java\javax.jar
   ```

4. Select “Output files” from the drop-down list and set the ”Output Directory” to:

   ```
   c:\iPlanet\ias6\ias\APPS
   ```

5. Choose OK and then save your Visual Cafe project.

   You may wish to save the “generic” project you have created. To customize it to debug a particular iPlanet Application Builder project, open the project in Visual Cafe and use the “File – Save As” command to save it under a different name.

6. Add the servlets and other Java classes located in your iPlanet Application Builder Project directory to the Symantec project using the Insert – Files Into Project... menu item. (Be sure these servlets have been registered by using the Build – Register – All Files menu item within Application Builder.)

   The first time you add a Servlet java file to your Visual Cafe project, make sure Visual Cafe has not changed your project type to "Servlet". Check the project options (Project – Options... menu item.) make sure the project type is still set to “Application - A program that requires java.exe to run.

Invoking iAS with a Visual Cafe Project

1. Stop iAS if it is running.

   - If iAS is running as a Windows system service, stop the service from the Windows Control panel.
   - If iAS was started using iPlanet Application Builder, use the Stop Server menu item in iAB’s Test menu to stop the service.

2. Restart iAS executive process manually by executing `kxs` from the command line in an MS-DOS window.

3. Within Symantec, choose the Project – Run in Debugger menu item or press the F5 key to launch the Java engine.

4. View the Messages window using the View – Messages menu item and look for a `ready: <port number>` message indicating that the server is ready to take requests.
5. Set at least one breakpoint or specify other debug options within Visual Cafe.
6. Open your browser to the index page of your application and exercise the servlets you want to debug.

Using Visual Cafe

You can use Visual Cafe with a wide range of debugging techniques. Several of the most common ways to use Visual Cafe are to

- set, view, disable, enable, and remove breakpoints
- step into, out of, or over code
- view stack information
- examine variables

The following sections identify how to perform these operations using Visual Cafe. For complete information, see your Visual Cafe documentation.

Manipulating Breakpoints

You can set, view, disable, enable, and remove breakpoints. To set a breakpoint, open the source file the contains the line of code you want to break on and perform one of the following actions:

- Click the line on which you want to break and from the Source menu choose Set Breakpoint
- Right-click the line on which you want to break and select Set Breakpoint
- Press F9 from the current line to set a breakpoint for the line (F9 toggles the breakpoint, removing it if already set)

When a breakpoint is set, Visual Cafe displays a diamond to the left of the line.

To view your breakpoints, choose Breakpoints from the View menu. Visual Cafe displays the list of breakpoints currently set in the project. Enabled breakpoints are identified with a check mark to the left of the location. You can uncheck a breakpoint to disable it. You can also enable or disable breakpoints from the Breakpoints menu. You can remove a breakpoint by highlighting it in the Breakpoints window and pressing the delete key.

Stepping through Code

Step operations are controlled by the Debug menu. This menu also allows you to stop and restart your application or continue execution from where it was stopped.

Viewing Stack Information

You can use the Call Stack window to view stack information. Choose Call Stack from the View menu to open the window. When this window is active, Visual Cafe presents a Calls menu, from which you can examine methods and variables on the stack as well as passed parameter values and their types.

Examining Variables

You can use the Variables window to examine the values of variables. Choose Variables from the View menu to open the window and view all the variables that are currently in context. You can click on a plus sign next to a variable to expand it if necessary. You can modify the contents of a variable only if the application is paused in the debugger.

Using iAB with Macromedia Dreamweaver™

If you have Dreamweaver™ installed, you can use it with iAB, either as the default editor or as an editor that you specify when you open an HTML file.

To specify Dreamweaver as an HTML editor

1. Choose Tools - Development Options...
2. Select the HTML tab
3. Specify the full path to `Dreamweaver.exe` as the External editor.
4. Check the "Double-click invokes external editor" if you want Dreamweaver to be the default editor; do not check this box if you want to use both the iAB editor and Dreamweaver.
5. Click OK to make the change.

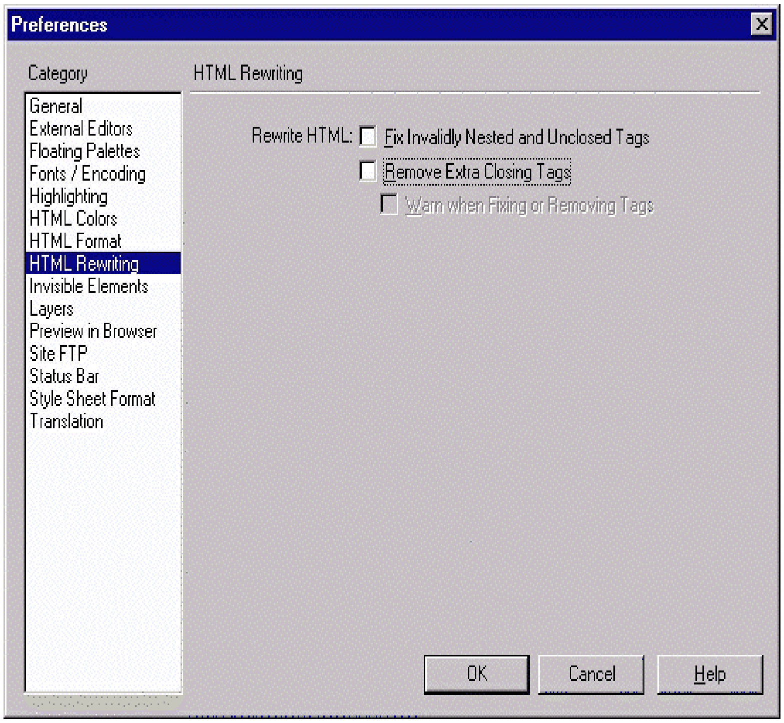
If you selected the “Double-click invokes external editor” box in iAB, Dreamweaver will open an HTML file when you double-click on the file within iAB. If you did not select this option, you will see an Open in Dreamweaver menu item when you right-click the mouse on the file you want to open; selecting this item opens the file in Dreamweaver.

Because you typically use the iAB editor when you specify data-bound properties, such as GX tags, you most likely will not want to use Dreamweaver by default.

To ensure that Dreamweaver does not modify your HTML automatically:

1. Open Dreamweaver.
2. Select Preferences from the Edit menu.
3. Select the HTML Rewriting category.
4. Uncheck all items under Rewrite HTML.

   Deselect “Fix Iinvalidly Nested and Unclosed Tags” and “Remove Extra Closing Tags.”


Preferences

Category: General, External Editors, Floating Palettes, Fonts / Encoding, Highlighting, HTML Colors, HTML Format, HTML Rewriting, Invisible Elements, Layers, Preview in Browser, Site FTP, Status Bar, Style Sheet Format, Translation

HTML Rewriting

Rewrite HTML: ☐ Fix Invalidly Nested and Unclosed Tags
☐ Remove Extra Closing Tags
☐ Warn when Fixing or Removing Tags

Using iAB with the Visual Cafe Java Editor

If you have Visual Cafe installed, you can use it with iAB when you open a Java file.

To specify Visual Cafe as a Java editor

1. Choose Tools - Development Options...
2. Select the Java tab.
3. Specify the full path to `VCafe.EXE` as the External editor.
4. Check the "Double-click invokes external editor" if you want Visual Cafe to be the default editor; do not check this box if you want to use both the iAB editor and Visual Cafe.
5. Click OK to make the change.

   If you cannot open the file using Visual Cafe, check that Visual Cafe's `bin` folder is at the start of your path.

Using iAB with the Visual Cafe Java Compiler

If you have Visual Cafe installed, you can use it with iAB when you compile a Java file.

To specify Visual Cafe as your compiler

1. Choose Tools - Development Options...
2. Select the Java tab.
3. Specify the full path to `sj.exe` as the Compile command.
4. Specify the compiler class path.
5. Click OK to make the change.

Appendix C

Developing Existing Netscape Application Server Applications with iPlanet Application Builder 6.0

This appendix describes how you can use applications that run under Netscape Application Server 2.1 or 4.0 with iPlanet Application Builder 6.0

- Migrating to Application Builder 6.0 Programming Model Programming Model
- Creating an Application Builder 6.0 Project
- Deploying to Application Server 6.0

Migrating to Application Builder 6.0 Programming Model

iPlanet Application Builder 6.0 projects work with the programming model defined by J2EE and the iPlanet Application Server 6.0. Therefore, individual source code and descriptor files need to be converted to reflect this programming model. See the Application Server *Migration Guide* for specific information on converting source files and descriptors to the iPlanet Application Server programming model.

Creating an Application Builder 6.0 Project

Once the migration procedures described in the Application Server *Migration Guide* have been followed, you can create a new project using Application Builder 6.0. Choosing the same project will save time in converting Java source file package names. Copy the converted files into the project directory and use the Application Builder's Add Files menu option to bring them into the new project. Similarly copy the static files (typically .html and image files) into a directory having the project directory's name under your web server's "document root" and these files to the project.

Finally, exit the Application Builder and copy the converted descriptor over the ones generated by the iPlanet Application Builder. This preserves any descriptor information applicable to the new programming model. Relaunch the Application Builder and compile and test the project. Keep in mind that due to changes in the Servlet, EJB, JSP and iAS APIs, manual changes to the source files will be necessary.

For more information about migration, see the iPlanet Application Server *Migration Guide*.

Deploying to Application Server 6.0

To deploy your application to the iPlanet Application Server 6.0 you can use the deployment facility provided in Application Builder 6.0.

If you still have an installed copy of the Application Builder 4.0, 3.0, or 2.x deployment of applications by these tools to the Application Server 6.0 is supported and will automatically deploy all necessary class files.

Application Builder 6.0 does not support deployment to versions of the Application Server earler than 6.0, nor can it download any applications deployed by Application Builder 3.0 or earlier.

Glossary

ACL Access Control List, a list of users or groups and their specified permissions. See *component ACL*, *general ACL*.

administration server A process in iPlanet Application Server that handles administrative tasks.

administrator See system administrator.

aggregate expression An expression in a query that summarizes values from one database column across several rows. You can use aggregate expressions to specify computed fields. The aggregate functions available depend on your database server, but those typically supported are `Min()`, `Max()`, `Count()`, `Avg()`, `Sum()`, `First()`, and `Last()`.

alias An alternate name. In a query, an alias is a name given to a database table, column, or computed field.

API Application Programmer Interface, a set of instructions that a computer program can use to communicate with other software or hardware that is designed to interpret that API.

applet A small application written in Java that runs in a web browser. Typically, applets are called by or embedded in web pages to provide special functionality. By contrast, a *servlet* is a small application that runs on a server.

application A computer program that performs a task or service for a user. Also see *web application*.

application event A named action that you register with the iAS registry. The event occurs either when a timer expires or when the event is called (triggered) from application code at run time. Typical uses for events include periodic backups, reconciling accounts at the end of the business day, or sending alert messages.

application flow The perceived progress of activity from page to page in a browser-oriented application.

application model The conceptual division of a software application into functional components.

application resource file Stores session information specific to your iPlanet Application Builder project. iPlanet Application Builder automatically creates a default base session, `AppResource.java`, which you can customize as you develop your application. This file stores code generated by iPlanet Application Builder wizards and other property editors. Specifically, `AppResource.java` stores session information specific to the project.

application server A program that runs an application in a client/server environment, executing the logic that makes up the application and acting as middleware between a web browser and a datasource.

application tier A conceptual division of an application:

- client tier: The user interface (UI). End users interact with client software (web browser) to use the application.
- server tier: The business logic and presentation logic that make up your application, defined in the application's components.
- data tier: The data access logic that enables your application to interact with a datasource.

AppLogic A iAS-specific cache responsible for completing a well-defined, modular task within a iPlanet Application Server application. In iAS 2.1, applications used AppLogics to perform actions such as handling form input, accessing data, or generating data used to populate HTML templates. This functionality is replaced with servlets and JSPs in iAS 4.0.

AppPath A iAS registry entry that contains the name of the directory where application files reside. This entry defines the top of a logical directory tree for the application, similarly to the document path in a web page. By default, AppPath contains the value `BasePath/APPS`, where `BasePath` is the base iAS directory. (BasePath is also a iAS variable.)

attribute Attributes are name-value pairs in a request object that can be set by servlets. Contrast with *parameter*. More generally, an attribute is a unit of metadata.

authentication The process of verifying a user-provided username and password.

banded report See grouped report.

base class See also superclass. A class from which another class is derived.

BasePath A iAS registry entry that contains the directory where iAS is installed, including the `iAS` subdirectory (other iPlanet products can also be installed in BasePath). BasePath is a building block for AppPath.

bean property file A text file containing EJB deployment information. The type of information is defined in `javax.ejb.DeploymentDescriptor`.

bean-managed transaction See *declarative transaction.*

binary large object (BLOB) A large block of bits that can be stored in a database. A BLOB is useful for storing any large piece of data, such as pictures or sounds, that does not need to be interpreted by the database.

browser See web browser.

browser events Actions that occur on the browser page, such as passing the cursor over a particular component, that can trigger actions specified by JavaScript objects on the page.

build project Compile all source files in the project that have been edited since the last time they were compiled.

business logic The implementation rules determined by an application's requirements.

business method Method that performs a single business task, such as querying a database or authenticating a user, in the course of business logic.

C++ server A process in iPlanet Application Server that runs and manages C++ objects.

cache See result cache.

cached rowset A `CachedRowSet` object permits you to retrieve data from a datasource, then detach from the datasource while you examine and modify the data. A cached rowset keeps track both of the original data retrieved, and any changes made to the data by your application. If the application attempts to update the original datasource, the rowset is reconnected to the datasource, and only those rows that have changed are merged back into the database.

callable statement A cache that encapsulates a database procedure or function call for databases that support returning result sets from stored procedures.

cell tag A type of GX markup tag that displays a dynamic data value.

class A named set of methods and member variables that define the characteristics of a particular type of object. The class defines what types of data and behavior are possible for this type of object.

class file A file that contains a compiled cache, usually with a `.class` extension. See also *class name, classpath.* Normally referred to in terms of its location in the filesystem, as in `...\com\myDomain\myPackage\myClass`.

class loader A Java component responsible for loading Java cachees, according to specific rules.

class name The name of a cache in the Java Virtual Machine. See also *class file, classpath.*

classpath The path that identifies a Java cache or package, in terms of its derivation from other classes or packages. See also *class file, class name*. For example, `com.myDomain.myPackage.myClass`.

client A computer or application that contacts and obtains data from a server on another computer. A client program is designed to work with one specific type of server. Or an entity that invokes a resource.

client contract A contract that determines the communication rules between a client and the EJB container, establishes a uniform development model for apps that use EJBs, and guarantees greater reuse of beans by standardizing the relationship with the client. See *Enterprise JavaBean (EJB).*

clean project Remove all object files from the project, leaving only source.

cluster A set of hosts running the same server software in tandem with each other.

co-locate Positioning a component in the same memory space as a related component in order avoid remote procedure calls and improve performance.

column A field in a database table.

commit Complete a transaction by sending the required commands to the database. See *transaction*.

compile To translate source code written by a programmer into object code that can run on a computer. A compiler is a program that performs this translation.

component Reusable objects that you can place on a page or template to perform a certain task. For example, an ImageLink component uses a GIF or JPG image as an anchor for an HTML hypertext link. The behavior and appearance of components are determined by their properties. See Appendix A, "Component Reference" for definitions of each component.

component ACL A property in a servlet or EJB configuration file that defines that defines the users or groups that may execute.

component contract A contract that establishes the relationship between an Enterprise JavaBean (EJB) and its container. See *Enterprise JavaBean (EJB).*

Component Object Model (COM) A specification that provides a standard way for objects and their clients to interact. COM specifies only how objects interact, not how applications are structured internally or how they are implemented.

computed field A field in a query that displays the result of an expression rather than stored data. The database engine recalculates the value each time it runs the query.

configuration The process of providing metadata for a component. Normally, the configuration for a specific component is kept in a file that is uploaded into the registry when the component executes.

connection A database connection is a communication link with a database or other data source.

constructor A method that instantiates a class.

container A process that executes and provides services for an EJB.

context, server A programmatic view of the state of the server, represented by an object.

cookie A variable that your application can send to a web browser to be stored there for a specified length of time. Each time a web browser views an HTML page in your application, the cookies from that browser are sent to the application. Cookies are domain-specific and can take advantage of the same web server security features as other data interchange between your application and the server. Thus, cookies are useful for privately exchanging data between your application and the web browser.

CORBA Common Object Request Broker Architecture, a standard architecture definition for object-oriented distributed computing.

custom property editor A dialog box that helps you determine the correct value for a complex property definition.

data access logic Business logic that involves interacting with a datasource.

database A generic term for Relational Database Management System (RDBMS). A software package that enables the creation and manipulation of large amounts of related, organized data.

database connection A database connection is a communication link with a database or other datasource. Components can create and manipulate several database connections simultaneously to access data.

database forms wizard A wizard that produces an HTML page with a data input form, a data model to establish database table relationships, a query based on the data model, a servlet to process the query, a JSP to display the results of the query, and an optional search form to enable user-initiated database queries.

data access logic Determine what types of back-end data sources (if any) the application accesses, such as a relational database or a legacy system.

data connection A logical connection between an application and a relational database.

data field One column of data in a RowSet.

data model An entity relationship (ER) diagram that specifies the data source tables and relationships used in your application.

data source A collection of data electronically stored within a relational database, legacy system, or object database.

default A value that is automatically assigned by the application when the user or programmer does not specify a value.

DELETE query A statement that specifies which data to delete from a database.

deploy To create a copy of all the files in a project on one or more servers, in such a way that one or more iPlanet Application Servers and optionally one or more web servers can run the application.

deployment descriptor An attribute that determines how and where an Enterprise JavaBean (EJB) is deployed. See *Enterprise JavaBean (EJB).*

design-time The behavior (in the test server) or appearance (in the iPlanet Application Builder windows) of an object when the application is being developed.

detail record The result of a secondary query, based on a master record.

Directory Server An LDAP server that is bundled with iPlanet Application Server. Every instance of iPlanet Application Server uses Directory Server to store shared server information, including information about users and groups.

distributable session A user session that is distributable among all servers in a cluster.

distributed computing A collection of computers linked together. Such systems can exist on a local area network (LAN), a wide area network (WAN), or the Internet. Distributed systems make several types of advanced computing systems possible, including client/server, multi-tier, and partitioned applications.

distributed transaction A single transaction that can apply to multiple heterogeneous databases that may reside on separate servers.

DLM See dynamically loadable module (DLM).

dockable window A window, such as the Project or Properties window, that has the ability to “snap into place” against the workspace border. When a window is attached in this way, it cannot be overlapped by other windows.

download project Copying application files from a server back to a development machine. Note that this does not re-create the development environment if the files were filtered during development.

drag and drop Clicking an object, holding the mouse button down while moving the cursor and the object to a destination (dragging), and then releasing the button to insert the object at the destination (dropping).

dynamically loadable module (DLM) A binary executable file that can be loaded while an application is running. In Windows NT or Win95 systems, DLM is another name for a Dynamic-Link Library (DLL). In UNIX systems, DLMs are implemented as ELF shared libraries.

Editor Beans Java objects which generate and maintain components.

dynamic reloading Updating and reloading a component without restarting the server. By default, servlet and JavaScript components can be dynamically reloaded.

e-commerce A term meaning electronic commerce, indicating business done over the Internet.

Enterprise JavaBean (EJB) A business logic component for applications in a multitiered, distributed architecture. EJBs conform to the Java EJB standard specifications, which defines beans in terms of their expected roles. An EJB encapsulates one or more application tasks or application objects, including data structures and the methods that operate on them. Typically they also take parameters and send back return values. EJBs always work within the context of a container, which serves as a link between the EJBs and the server that hosts them. See also *container*, *session EJB*, and *entity EJB*.

entity EJB An entity Enterprise JavaBean (EJB) relates to physical data, such as a row in a database. Entity beans are long-lived, because they are tied to persistent data. Entity beans are always transactional and multiuser aware. Also see *session EJB*.

ER diagram Describes the attributes of database entities and the relationships among them.

event Named actions that you register with the iPlanet Application Server. The event occurs either when a timer expires or when the event is called from application code at run time. Typical uses for events include periodic backups, reconciling accounts at the end of the business day, or sending alert messages.

event handler JavaScript object on an HTML page or JavaServer Page that handles a browser event at run time.

executive server Process in iPlanet Application Server that handles executive functions such as load balancing and process management.

failover recovery A process whereby a bean can transparently survive a server crash.

field The smallest identifiable part of a table in a database. A field is the intersection of a row and a column. A unit of data in a result set. Each field in a result set has a name, which corresponds to either a database column or an expression. Each field contains a single data value.

filtering Removing development-oriented information from files while deploying.

finder method Method which enables clients to look up a bean or a collection of beans in a globally available directory. See *Enterprise JavaBean (EJB)*.

flat query A query that produces a result set that is not divided into levels or groups. The result set of a flat query is like a table.

floating toolbar The state of a toolbar when it appears unattached in the center of the workspace, as opposed to being docked along the workspace borders. Floating toolbars have title bars to identify them.

form action handler A specially defined method in a servlet or AppLogic that performs an action based on a named button on a form.

format mask A mask applied to data to specifically tailor its format for display. Options include numeric formats (integer, percentage), and custom date formats. Also known as display format.

general ACL A named list in the Directory Server that relates a user or group with one or more permissions. This list can be defined and accessed arbitrarily to record any set of permissions.

generated code JavaScript code generated by Editor Beans which should not be edited.

generic servlet A servlet that extends `javax.servlet.GenericServlet`. Generic servlets are protocol independent, meaning that they contain no inherent support for HTTP or any other transport protocol. Contrast with *HTTP servlet*.

global database connection A database connection available to multiple component. Requires a resource manager.

global transaction A transaction that is managed and coordinated by a transaction manager and can span multiple databases and processes. The transaction manager typically uses the wizard to interact with the database backends. Also see *local transaction*.

globally unique identifier (GUID) A unique number that identifies a servlet object and is used to request that iPlanet Application Server runs that servlet.

group A set of rows in a result set that have one or more field values in common or a group of users that are related in some way, maintained by a local system administrator. See also *user*, *role*.

grouped report A report that shows records in logical groups, such as sales grouped by geographic region, and can show summary data for each group.

GUID 128-bit hexadecimal number, guaranteed to be globally unique, used to identify components in a iAS application.

GX markup tag A special type of syntax used in templates to indicate where dynamic data is to be merged with the template. A GX tag is made up of two tags, <GX ...> and </GX>, and the text between them. Some GX tags are represented with % rather than < or >, as in %GX TYPE="cell" ...% %/GX%. This is equivalent to <GX TYPE="cell" ...> </GX> at run time.

.gxm file A file that keeps track of all files belonging to a project. Also called a project file.

GXML template A definition for a dynamically generated set of output data. Data retrieved from a database or other data source at run time is sent back to the client in a self-describing stream of output.

.gxr file A file containing information that allows .java files and other files in a project to be registered with the iPlanet Application Server. Also called a registration.

handle The vertical strip on the left side of a toolbar by which you can drag the toolbar.

HTML Hypertext Markup Language. A coding markup language used to create documents that can be displayed by web browsers. Each block of text is surrounded by codes that indicate the nature of the text.

HTML page A page coded in HTML and intended for display in a web browser.

HTTP A protocol for communicating hypertext documents across the Internet.

HTTP servlet A servlet that extends `javax.servlet.HttpServlet`. These servlets have built-in support for the HTTP protocol. Contrast with *generic servlet*.

hyperlink A word or phrase that the user can click to display another page in an online document.

identity An instance of `java.security.Identity`, a security object that contains information about a specific entity, such as a user or a group.

IIOP Internet Inter-Orb Protocol. Transport protocol for RMI clients and servers, based on CORBA.

image URL Source code for an image. The URL can be relative (local server) or absolute (local or remote server). The URL can be determined dynamically if the component that requires it is in a template.

include tag A type of GX markup tag that displays HTML output created by evaluating another template.

inheritance A mechanism in which a subclass automatically includes the method and variable definitions of its superclass. A programmer can change or add to the inherited characteristics of a subclass without affecting the superclass.

input validation The set of rules which defines a variable.

input wizard A wizard that produces a static HTML page containing an input form, a servlet to process the form, a JSP to display the results, and optionally a query to retrieve data from a data source.

INSERT query A statement that specifies which data to add to a database.

instance An object that is based on a particular cache. Each instance of the class is a distinct object, with its own variable values and state. However, all instances of a class share the variable and method definitions specified in that class.

instantiation The process of allocating memory for an object at run time. See *instance.*

interface Description of the services provided by an object. An interface defines a set of functions, called methods, and includes no implementation code. An interface, like a cache, defines the characteristics of a particular type of object. However, unlike a class, an interface is always abstract. A class is instantiated to form an object, but an interface is implemented by an object to provide it with a set of services. Contrast with *cache.*

isolation level (JDBC) Sets the level at which the datasource connection makes transactional changes visible to calling objects such as ResultSets.

jar file contract A contract that specifies what information must be in the Enterprise JavaBean (EJB)'s package (`jar` file). See *Enterprise JavaBean (EJB).*

JavaBean A discrete, reusable Java object.

Java server Process in iPlanet Application Server that runs and manages Java objects.

JavaScript A language that can run as a script in an HTML page, allowing screen actions outside the scope of HTML, including responses to browser events.

JavaServer Page (JSP) A text page written using a combination of HTML or XML tags, JSP tags, and Java code. JSPs combine the layout capabilities of a standard browser page with the power of a programming language.

JDBC Java Database Connectivity administrators. A standards-based set of cachees and interfaces that enable developers to create data-aware components. JDBC implements methods for connecting to and interacting with datasources in a platform- and vendor-independent way.

JNDI Java Naming and Directory Interface. JNDI provides a uniform, platform-independent way for applications to find and access remote services over a network. iAS supports JNDI lookups for datasources and Enterprise JavaBean (EJB) components.

kas See *administration server*.

kcs See *C++ server*.

kjs See *Java server*.

kxs See *executive server*.

layout view An HTML editor window display that shows the HTML page similarly to how it appears in a browser at runtime.

LDAP Lightweight Directory Access Protocol. LDAP is an open directory access protocol that runs over TCP/IP. It is scalable to a global size and millions of entries. Using Directory Server, a provided LDAP server, you can store all of your enterprise's information in a single, centralized repository of directory information that any application resource file can access via the network.

link URL A target for a hypertext link. These include static pages, JSPs, servlets, and other web sites. The URL can be relative (local server) or absolute (local or remote server). The URL can be determined dynamically if the component that requires it is in a template.

listing See tabular report.

load balancing A technique for distributing the user load evenly among multiple servers in a cluster. Also see *sticky load balancing*.

local database connection The transaction context in a local connection is not distributed across processes or across datasources; it is local to the current process and to the current datasource.

local session A user session that is only visible to one server.

local transaction A transaction that is native to one database and is restricted within a single process. Local transactions can work against only a single backend. Local transactions are typically demarcated using JDBC APIs. Also see *global transaction*.

login wizard A wizard that produces a static HTML page containing a login form, a servlet to verify a user name and password, and a JSP to display the results of a successful login.

master record The primary target of a query.

member A variable or method declared in a class.

member variable A variable with the following characteristics:

- The variable is declared inside a class declaration.
- A member variable specifies a piece of data that can be stored by an object instantiated from that class.

memory cache A iAS feature that enables a servlet to cache its results for a specific duration in order to improve performance. Subsequent calls to that servlet within the duration are given the cached results so that the servlet does not have to execute again.

metadata Information about a component, such as its name, and specifications for its behavior.

method A function with the following characteristics:

- The method is declared inside a class or interface.
- A method specifies an action that can be performed by an object instantiated from that class.

metadata Represents information that is passed into the runtime's AppResource constructor.

iAS registry A collection of application metadata, organized in a tree, that is continually available in active memory or on a readily-accessible Directory Server.

iASRowSet

A RowSet object that incorporates iAS extensions. The `iASRowSet` class is a subclass of `ResultSet`.

iPlanet Application Server Foundation Class Library A set of interfaces and classes provided by Netscape Communications Corporation that can be used to develop object-oriented iPlanet Application Server applications. The classes in the iPlanet Application Server Foundation Class Library define many types of objects you can include in iPlanet Application Server applications, such as servlet objects, data connections, queries, and result sets.

NTV A Name-Type-Value format, used in iAS for servlet and application configuration files.

object A programmed entity with the following characteristics:

- An object embodies both data and behavior.
- Objects come into existence at run time through the process of instantiation.
- Each object is based on a definition, which is called a class.

Many parts of a iPlanet Application Server application, such as servlets, queries, and result sets, are objects.

object-oriented programming A method for writing programs using classes, not algorithms, as the fundamental building blocks. At run time, the classes give rise to objects, which perform the tasks of the application.

OCL See Open Client Library (OCL).

ODBC Open Database Connectivity (ODBC)

online application server A server that stores, manages, and executes dynamic Internet and intranet applications. An online application server is specifically designed to run such applications quickly and efficiently. iPlanet Application Server is an online application server.

Open Database Connectivity (ODBC) A standard protocol used by many database vendors to provide an interface to outside applications. iPlanet Application Server applications can interact with databases that comply with ODBC 1.0 and 2.0.

outline view An HTML editor window display that shows the structural relationships between HTML tags on the page.

override To write new code that replaces the default code of an inherited method.

palette A window containing components that you can drag and drop into HTML files to create pages and templates.

palette pages Individual sections of the palette that contain similar elements, accessed by tabs at the bottom of the palette window. If a page is locked, it can not be deleted and its resident components can not be removed.

parameter The data passed between methods, servlet objects, and other program code. A placeholder for dynamic data that is passed into a prepared database command at run time. Name-value pairs sent from the client, including form field data, HTTP header information, etc., and encapsulated in a request object. Contrast with attribute.

package A collection of related cachees that are literally packaged together in a Java archive (`.jar`) file.

passivation A method of releasing an EJB's resources without destroying the bean. In this way, a bean is made to be persistent, and can be recalled without the overhead of instantiation. See *Enterprise JavaBean (EJB)*.

permission A set of privileges granted or denied to a user or group. See also *ACL*.

persistent Refers to the creation and maintenance of a bean throughout the lifetime of the application. In iAS 4.0, beans are responsible for their own persistence, called *bean-managed persistence*. Opposite of *transient*.

pooling Providing a number of preconfigured resources to improve performance. If a resource is pooled, a component can use an existing instance from the pool rather than instantiating a new one. In iAS, database connections, servlet instances, and Enterprise JavaBean (EJB) instances can all be pooled.

prepared command A database command (in SQL) that is precompiled to make repeated execution more efficient. Prepared commands can contain parameters. A prepared statement contains one or more prepared commands.

prepared statement A cache that encapsulates a query, update, or insert statement that is used repeatedly to fetch data. A prepared statement contains one or more prepared command.

presentation layout Creating and formatting page content.

presentation layer The set of classes (servlets) and related files that controls the interface an application presents to its users.

presentation layout Creating and formatting page content.

presentation logic Activities that create a page in an application, including processing a request, generating content in response, and formatting the page for the client.

primary key class name A variable that specifies the fully qualified class name of a bean's primary key. Used for JNDI lookups.

process A sequence of execution in an active program. A process is made up of one or more threads.

programmatic security Controlling security explicitly in code rather than allowing the component's container (i.e., a bean's container or a servlet engine) to handle it. Opposite of declarative security.

programmatic transaction Controlling a transaction explicitly in code rather than allowing an Enterprise JavaBean (EJB)'s container to handle it. Opposite of declarative transaction.

project A collection of related files that, when deployed, constitute a web application.

project file See .gxm file.

project map A window that displays file dependencies. A project map is useful for visually representing the page flow of an application.

project window A directory listing of files in a project. Files can be grouped by folder or listed alphabetically.

properties window A window showing the properties for a selected object.

property Name-value pairs that indicate how an object behaves or appears. For example, a Name property might contain a name that identifies the object to other objects, while a Background Color property defines a background color for the object. Or a single attribute that defines the behavior of an application component.

property definition Value associated with a given property that, together with the other properties for a given object, determines the object's appearance and/or behavior.

query A statement that specifies which data to retrieve from a database. Typically, the results of a query are displayed in a report.

query file A query file is a file that contains the specification for a flat or hierarchical query. Query files are useful for running legacy SQL SELECT statements. You can also use query files to write new queries.

rebuild project Remove all object files (clean project) and compile all source files in the project.

register Register the Java methods with the Java Virtual Machine running on the application server.

registered servers Servers registered to iPlanet Application Builder for the purpose of deployment. You can not deploy to a server until it is registered.

registration The process of informing iPlanet Application Server of the existence of a servlet object, code module, or security information. Or, the process by which iAS gains access to a servlet, Enterprise JavaBean (EJB), and other application resource, so named because it involves placing entries in the iAS registry for each item.

registry file See .gxr file.

relationship A named connection between data source tables.

remote interface Describes how clients can call a Enterprise JavaBean (EJB)'s methods. See *Enterprise JavaBean (EJB)*.

remote procedure call (RPC) A mechanism for accessing a remote object or service.

replace tag A type of GX markup tag that substitutes a dynamic data value for a specified string.

report A formatted presentation of data. In a iPlanet Application Server application, a report is an HTML page presented to the user in response to a request for information. Servlet objects create reports by combining hierarchical result sets and JavaServer Pages.

request A message from a client to a server, asking for data or another service.

- In a iPlanet Application Server application, a request is a message that causes an servlet object to run on the iPlanet Application Server. A request uses a unique name or globally unique identifier (GUID) to identify the proper servlet object to handle the request. The request can include parameters to be passed to the servlet object. Requests can come from clients, servlet objects, or other code.

response A generated HTML page that contains a logical answer to the request, and sets up the next interaction.

response object An object that references the calling client and provides methods for generating output for the client.

result cache Storage in iPlanet Application Server that holds the output from an servlet object so that the output can be accessed repeatedly without the necessity of running the servlet object again.

results wizard A wizard that produces a servlet to retrieve data, optionally accessing a query to retrieve data from a data source, and an HTML template to display the results.

reusable component A component created so that it can be used in more than one capacity, i.e., by more than one resource or application.

RMI Remote Method Invocation (RMI), a Java standard set of administrators that enable developers to write remote interfaces that can pass objects to remote processes.

role A functional grouping of subjects in an application, represented by one or more groups in a deployed environment. See also *user, group.*

rollback Cancel a transaction. See *transaction*.

row One single data record that contains values for each column in a table.

RowSet An object that encapsulates a set of rows retrieved from a database or other source of tabular data. RowSet extends the `java.sql.ResultSet` interface, enabling a ResultSet to act as a JavaBeans component.

runtime The behavior (in the server) or appearance (in a browser) of an object when the application runs.

select distinct (query) A query type that retrieves only the unique instances of the requested target search items.

SELECT query A statement that specifies which data to retrieve from a database, as specified by your data model.

serializable An object is serializable if it can be deconstructed and reconstructed, which enables it to be stored or distributed among multiple servers.

sequence A sequential number generator which exists in a database. Some database vendors refer to a sequence as a serial, identity, or autoincrement. A sequence is useful for generating transaction-safe numbers for database transaction applications.

server A computer or software package that provides a specific kind of service to client software running on other computers. A server is designed to communicate with a specific type of client software.

servlet An instance of the `Servlet` cache. A servlet is a reusable application that runs on a server. In iAS, a servlet acts as the central dispatcher for each interaction in your application by performing presentation logic, invoking business logic, and invoking or performing presentation layout.

servlet engine An internal object that handles all servlet metafunctions. Collectively, a set of processes that provide services for a servlet, including instantiation and execution.

servlet runner Part of the servlet engine that invokes a servlet with a request object and a response object. See *servlet engine.*

session A continuous series of interactions between a user and a iPlanet Application Server application.

session accessors Any method that gets or sets a session variable.

session cookie A cookie that is returned to the client containing a user session identifier.

session EJB A session Enterprise JavaBean (EJB) relates to a unit of work, such as a request for data. Session beans are short lived—the lifespan of the client request is the same as the lifespan of the session bean. Session beans can be stateless or stateful, and they can be transaction aware. See *stateful session EJB* and *stateless session EJB.* Also see *entity EJB.*

session timeout A specified duration after which iAS can invalidate a user session. See *user session*.

session validation The set of rules that guarantees that the application is in a valid state to perform the functionality of a specific servlet.

session variables Used by many servlets in an application to store and access data that is shared throughout the application.

source view An HTML editor window display that shows the source code for the page.

SQL Structured Query Language (SQL) is a language commonly used in relational database applications. SQL2 and SQL3 designate versions of the language.

standard query A query that produces a result set that is not divided into levels or groups. The result set of a standard query is like a table.

state **1.** The circumstances or condition of an entity at any given time. **2.** A distributed data storage mechanism which you can use to store the state of an application using the iAS feature interface IState2.

stateful session EJB An Enterprise JavaBean (EJB) that represents a session with a particular client and which automatically maintains state across multiple client-invoked methods.

stateless session EJB An Enterprise JavaBean (EJB) that represents a stateless service. A stateless session bean is completely transient and encapsulates a temporary piece of business logic needed by a specific client for a limited time span.

sticky cookie A cookie that is returned to the client to force it to always connect to the same executive server process.

sticky load balancing A method of load balancing where an initial client request is load balanced, but subsequent requests are directed to the same process as the initial request. Also see *load balancing*.

stored procedure A block of statements written in SQL and stored in a database. You can use stored procedures to perform any type of database operation, such as modifying, inserting, or deleting records. The use of stored procedures improves database performance by reducing the amount of information that is sent over a network.

streaming A technique for managing how data is communicated via HTTP. When results are streamed, the first portion of the data is available for use immediately. When results are not streamed, the whole result must be received before any part of it can be used. Streaming provides a way to allow large amounts of data to be returned in a more efficient way, increasing the perceived performance of the application.

subclass A class that is derived from and is a special case of another class, called a base class or superclass.

superclass A class from which another class, called a subclass, is derived. See also base class.

system administrator The person who is responsible for installing and maintaining iPlanet Application Server software and for deploying production iPlanet Application Server applications.

table A named group of related data in rows and columns in a database.

tabular report A report, sometimes called a listing, that prints all the records retrieved from the database.

tag See GX markup tag or HTML.

target window Name of the window or frame that displays the results of a hypertext link.

template engine The part of the server responsible for taking JavaServer Pages and merging them with the data from a servlet.

template map An object that maps fields in a JSP to the data used to replace those fields. With a template map, you can assign values to special placeholders that will be evaluated at run time. You can also use a template map to link column names in a table to field names that you have used in a JSP. A template map allows your application to use the same JSP file with data from different data sources.

test server A version of the iPlanet Application Server that runs for local testing purposes.

thread A sequence of execution inside a process. A process may allow many simultaneous threads, in which case it is multithreaded. If a process executes each thread sequentially, it is single-threaded.

tile tag A type of GX markup tag which repeats the tags and text nested within it. Tile can be used in two ways: repeating a fixed number of times, or repeating for each row in a result set.

tooltip A word or phrase that appears whenever you briefly place the mouse over a toolbar button. Tool tips are useful reminders of a button’s purpose.

transaction A set of database commands that succeed or fail as a group. All the commands involved must succeed for the entire transaction to succeed.

transaction context A transaction's scope, either local or global. See *local transaction*, *global transaction*.

transaction manager Object that controls a global transaction, normally using the XA protocol. See *global transactions*.

transient A resource that is released when it is not being used. Opposite of *persistent*.

trigger A trigger is a stored block of SQL or PL/SQL statements that is associated with a table, runs in response to an INSERT, UPDATE, or DELETE operation, and runs only under certain specified conditions.

update query A statement that specifies which data to modify within a database.

URI Universal Resource Identifier, describes specific resource at a domain. Locally described as a subset of a base directory, so that `/ham/burger` is the base directory and a URI specifies `toppings/cheese.html`. A corresponding URL would be `http://`*`domain`*`:`*`port`*`/toppings/cheese.html`.

URL Uniform Resource Locator. An address that uniquely identifies an HTML page or other resource. A web browser uses URLs to specify which pages to display. A URL describes a transport protocol (e.g. HTTP, FTP), a domain (e.g. `www.my-domain.com`), and optionally a update query.

user A person who uses your application. Programmatically, a user name, password, and set of attributes that enables an application to recognize a client. See also *group, role*.

user interface (UI) The pages that define what a user sees and with which a user interacts in a web application.

user session A series of user-application interactions that are tracked by the server. Sessions maintain user state, persistent objects, and identity authentication.

user validation Validation written by the programmer.

validation A method for ensuring that the contents of a form field are within certain parameters. If a user enters data outside the parameters, a dialog box appears notifying them of the error and what to do to correct it. A iAS feature that enables validity checking of some types of form input.

variable A named storage location for data that can be modified while a program is running. Each variable has a unique name that identifies it within its scope. Each variable can contain a certain type of data.

versioning See *dynamic reloading.*

web application A computer program that uses the World Wide Web for connectivity and user interface (UI). A user connects to and runs a web application by using a web browser on any platform. The user interface of the application is the HTML pages displayed by the browser. The application itself runs on a web page and/or application resource file.

web browser Software that is used to view resources on the World Wide Web, such as web pages coded in HTML or XML.

web connector plug-in An extension to a web page that enables it to communicate with a iPlanet Application Server.

web page See HTML page.

web server A host that stores and manages HTML pages and web applications. The web server responds to user requests from web browsers.

wizard A code generator that provides a framework for creating the most commonly-used types of application development components.

World Wide Web A network of many computers linked together by their ability to understand the HyperText Transfer Protocol (HTTP). Two types of computers make up the Web: clients and servers. Clients are computers with web browsers installed on them. Servers are computers that store and manage the information requested by the clients.

workspace The main window of iPlanet Application Builder. The workspace contains the windows, toolbars, and dialog boxes that constitute the user interface.

XA protocol A database industry standard protocol for distributed transactions.

XML XML, the Extensible Markup Language, uses HTML-style tags to identify the kinds of information used in documents as well as to format documents.

Index