Migration Guide

iPlanet™ Application Server

Version 6.0

Contents

Preface

This guide describes how to migrate applications from Netscape Application Server versions 2.1 and 4.0 to iPlanet Application Server 6.0. In addition, this guide includes information for migrating Net Dynamics applications.

This preface contains the following sections:

- Using the Documentation
- How This Guide Is Organized
- Documentation Conventions
- Related Information

Using the Documentation

The following table lists the tasks and concepts that are described in the iPlanet Application Server manuals and *Release Notes*. If you are trying to accomplish a specific task or learn more about a specific concept, refer to the appropriate manual.

Note that the printed manuals are also available online in PDF and HTML format, at: `http://docs.iplanet.com/docs/manuals/ias.html`.

For information about	See the following	Shipped with
Late-breaking information about the software and the documentation	*Release Notes*	iPlanet Application Server 6.0
Installing iPlanet Application Server and its various components (Web Connector plug-in, iPlanet Application Server Administrator), and configuring the sample applications	*Installation Guide*	iPlanet Application Server 6.0

For information about	See the following	Shipped with
Administering one or more application servers using the iPlanet Application Server Administrator Tool to perform the following tasks: • Monitoring and logging server activity • Implementing security for iPlanet Application Server • Enabling high availability of server resources • Configuring web-connector plugin • Administering database connectivity • Administering transactions • Configuring multiple servers • Administering multiple-server applications • Load balancing servers • Managing distributed data synchronization • Setting up iPlanet Application Server for development	*Administrator's Guide*	iPlanet Application Server 6.0
Migrating your applications to the new iPlanet Application Server 6.0 programming model from the Netscape Application Server version 2.1, including a sample migration of an Online Bank application provided with iPlanet Application Server	*Migration Guide*	iPlanet Application Server 6.0

For information about	See the following	Shipped with
Creating iPlanet Application Server 6.0 applications that follow the open Java standards model (Servlets, EJBs, JSPs, and JDBC), by performing the following tasks:	*Developer's Guide (Java)*	iPlanet Application Server 6.0
• 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		
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)*	iPlanet Application Server 6.0
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

How This Guide Is Organized

This guide is organized into five chapters, as follows:

- Chapter 1, Migration Overview

- Chapter 2, Running NAS 2.1 Applications

- Chapter 3, Migrating NAS 2.1 Applications

- Chapter 4, Running NAS 4.0 Applications

- Chapter 5, Running NetDynamics Applications

In addition, there is a complete code walkthrough of an example migration using the Online Bank sample application from NAS 4.0 to iPlanet Application Server 6.0. This code example is also available online. Check the iPlanet web site support area for more migration details.

Documentation Conventions

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

This guide uses URLs of the form:

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

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

This guide uses the following font conventions:

- The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.

- *Italic* type is used for book titles, variables and placeholders, and words used in the literal sense.

Related Information

Specifications related to the iPlanet Application Server 6.0 programming model are provided in the docs directory on your installation CD. However, always refer to the online documentation first as this may have been updated since you have received the product.

The official specifications are maintained at the following URLs. Note that these sites do not necessarily contain the versions of these specifications that are supported by iPlanet Application Server.

For information about	See the following
Servlets	`http://java.sun.com/products/servlet`

For information about	See the following
JavaServer Pages (JSPs)	`http://java.sun.com/products/jsp`
Enterprise JavaBeans (EJBs)	`http://java.sun.com/products/ejb`
Java Naming and Directory Interface (JNDI)	`http://java.sun.com/products/jndi`
Java Database Connectivity (JDBC)	`http://java.sun.com/products/jdbc`

Additionally, the he following resources are available:

Programming with Servlets and JSPs

- *Java Servlet Programming*, by Jason Hunter with William Crawford, O'Reilly Publishing

- *Java Threads, 2nd Edition*, by Scott Oaks & Henry Wong, O'Reilly Publishing

- The web site *http://www.servletcentral.com*

Programming with EJBs

- *Enterprise JavaBeans*, by Richard Monson-Haefel, O'Reilly Publishing

- The web site *http://ejbhome.iona.com*

Programming with JDBC

- *Database Programming with JDBC and Java*, by George Reese, O'Reilly Publishing

- *JDBC*, by Graham Hamilton, Rick Cattell, Maydene Fisher

Related Information

Migration Overview

This chapter introduces the iPlanet Application Server 6.0 programming model and compares it to both the NAS 4.0 programming model and the NAS 2.1 programming model. It also describes the basics of migrating NAS applications to the new iPlanet Application Server 6.0 model. Finally, this chapter describes the procedures required for redeploying iPlanet Application Server 6.0 SP1 applications to iPlanet Application Server 6.0 SP3 using the new `redeploy` command utility.

The new iPlanet Application Server 6.0 programming model is for Java applications only. C++ applications continue to use the NAS 2.1 model. Note the following compatibility issues:

- iPlanet Application Server 6.0 is backward compatible with NAS 2.1 applications. NAS 2.1 applications can run on iPlanet Application Server 6.0 without code alteration.

- iPlanet Application Server 6.0 is compatible with NAS 4.0 applications with conversion to the J2EE standard. NAS 4.0 applications do require some conversion.

This chapter describes the following topics:

- The New J2EE Programming Model

- Migrating NAS 2.1 Applications to iPlanet Application Server 6.0

- Migrating NAS 4.0 Applications to iPlanet Application Server 6.0

- Redeploying iPlanet Application Server 6.0 SP1 Applications

The New J2EE Programming Model

iPlanet Application Server 6.0 is Java 2 Platform, Enterprise Edition (J2EE) specification version 1.2 compliant and is based on standards developed by the Java community, namely: servlets, JavaServer Pages, and Enterprise JavaBeans. This is in contrast to the proprietary AppLogic-based programming model used in NAS 2.1. NAS 4.0 is based on the J2EE programming model but uses earlier versions of the standards.

Application flow is similar between the iPlanet 6.0 model and the previous 4.0 and 2.1 models. Each user interaction is handled by one (or more) application components that process the inputs, perform business logic functions, interact with a database, and provide an output page that answers the input and sets up the next user interaction. The 6.0 model, like the 4.0 model, is more modular and segregates activities into more discrete components.

The new programming model describes three tiers of application logic, each of which is represented by a set of components or APIs. These tiers are described in the following table:

Programming Tier	NAS 2.1 component	NAS 4.0 component	iPlanet Application Server 6.0 component	Description
Presentation Logic	AppLogic	Java servlet and proprietary standards	Java servlet	Controls the application's interface to the user by processing requests, generating content in response, formatting and delivering that content back to the user. In 6.0, servlets process incoming requests and orchestrate the response. Business logic is normally off-loaded to EJBs, and output is usually off-loaded to JSPs.
Presentation Layout (part of Presentation Logic)	HTML template	JavaServer Page (JSP) and proprietary standards	JavaServer Page (JSP)	Controls the appearance of each page. Part of the presentation logic, usually handled by JavaServer Pages. JSPs are HTML pages that contain embedded Java, and thus are much more versatile and powerful than 2.1 HTML templates.

Programming Tier	NAS 2.1 component	NAS 4.0 component	iPlanet Application Server 6.0 component	Description
Business Logic	AppLogic	Enterprise JavaBeans (EJBs) and proprietary standards	Enterprise JavaBeans (EJBs)	Controls business logic. EJBs enable business logic to be persistent across calls, offer improved caching, and are designed to work closely with JDBC for database transactions.
Data Access Logic	DAE	JDBC and proprietary standards	JDBC	Controls database storage and retrieval. The JDBC API is available to all Java components, as are all APIs, though database transactions are usually controlled by EJBs in the 6.0 model.

This section includes the following additional topics:

- Component Modularity and Flexibility

- Presentation Logic and Layout

- Business Logic

- Data Access Logic

Component Modularity and Flexibility

The terms "normally" and "usually" appear frequently in this document and in the *Developer's Guide (Java)* with regard to the roles of iPlanet Application Server 6.0 components. Since servlets, JSPs, and EJBs all reside within the same virtual machine and are all Java objects, they share a flexibility that allows each task to be addressed by more than one component. There are no hard and fast rules specifying which tasks are appropriate for which components. For example, an entire complex application could be written using only JSPs, or only servlets.

However, the components are designed to work together in a modular way, taking advantage of the strengths of each component. For example, it is more cumbersome to perform layout tasks in a servlet, but JSPs (as HTML pages) are highly suitable for layout tasks. Alternatively, presentation logic is compact and elegant in a servlet.

The segregation and order of components describes a powerful application model that runs well in a distributed environment. Choose components that perform the tasks you need, using the programming tiers described here as a guideline.

Presentation Logic and Layout

Presentation logic describes the flow of an application from the perspective of each user interaction: request processing, followed by content generation and delivery. The goal of presentation logic is to create a logical answer to a request, and to prompt for another request. The goal of presentation layout is to display the content of this answer in a predetermined format. Application functions such as user sessions, security and user authentication, and input validation are also handled by the presentation logic.

In short, presentation logic involves everything related to the application's interface with the user.

In the NAS 2.1 programming model, presentation logic was controlled by the AppLogic class, while layout was handled by an HTML template. At run-time, AppLogic objects provided output to populate the template.

In the iPlanet Application Server 6.0 programming model, presentation logic is usually handled by a Java servlet. Layout is usually handled by a JSP. At runtime, the servlet uses a JSP to format the content generated by the business logic.

The two major alternatives to this basic model are as follows:

- Handle all presentation logic and layout for a given interaction in a JSP. This can be an easy way to control an interaction that has no business logic and little to process from the previous interaction. For example, the "front page" for an application often requires no processing at all.

- Handle all presentation logic and layout in a servlet. This can be efficient for interactions that have very little layout. For example, a simple database report might just list the rows retrieved from a database query. It doesn't make sense to incur the overhead of a JSP call when the page can be simply output from a servlet.

Business Logic

Business logic describes the activities that involve the generation of specific content: storing and retrieving data, and performing computations on that data. The goal of business logic is to perform the activities that generate or determine answers to questions posed by the presentation logic.

In short, business logic involves the content provided by and generated for the application.

In the NAS 2.1 programming model, business logic was controlled by the same AppLogic that handled the presentation logic for a given user interaction.

In the iPlanet Application Server 6.0 programming model, business logic is usually handled by one or more Enterprise JavaBeans (EJBs), which control database transactions and encapsulate the results. EJBs are powerful, reusable components that empower applications with a great deal of flexibility, since EJBs can be invoked or inspected from any other object and can be made to be persistent.

One alternative to this model is to handle business logic in the presentation logic (servlets and/or JSPs), much the same way that AppLogics handled business logic. This can be efficient for short, directed business events such as specific directory requests, but this approach lacks the flexibility and power that EJBs bring to the programming model.

Data Access Logic

Data access logic describes transactions with a database or directory server. The goal of data access logic is to provide an interface between an application and the set of data that concerns it. Data access is normally performed as a function of business logic.

In short, data access logic involves the storage and retrieval of the content collected or generated by business logic.

In the NAS 2.1 programming model, data access logic was controlled by calls made from an AppLogic using APIs from several classes and interfaces, including the `DataSet`, `DBDataSet`, and `DBStoredProcedure` classes and the `ICallableStmt`, `IColumn`, `IDataConn`, `IDataConnSet`, `IHierQuery`, `IHierResultSet`, `IListDataSet`, `IPreparedQuery`, `IQuery`, `IResultSet`, `ITable`, `ITrans`, and `IValList` interfaces.

In the iPlanet Application Server 6.0 programming model, data access logic is handled by the JDBC standard set of APIs. The previous APIs are all deprecated in iPlanet Application Server 6.0.

Migrating NAS 2.1 Applications to iPlanet Application Server 6.0

Migration involves altering an application written for the NAS 2.1 programming model so that it conforms to the iPlanet Application Server 6.0 programming model. There are three approaches to this process, each of which is covered in this document:

- **No migration.** This approach involves no actions by the developer and depends solely on backward-compatible support by the server. This is an acceptable approach if you do not want to take advantage of the flexibility and power that the new standards-based model provides, although many of the APIs supported in NAS 2.1 are now deprecated and may not be supported in future releases.

  Backward-compatibility is described in Chapter 2, "Running NAS 2.1 Applications."

- **Partial migration.** In this approach, part of the application conforms to the new programming model, while the rest relies on backward-compatibility. This enables developers to migrate one portion of an application at a time (for example, one level of interaction with a user, or one programming tier) while still retaining the portions of the application that are known and tested.

  iPlanet Application Server 6.0 supports partial migration by providing "glue" between the old components and the new components. This support is described in "Partial Component Migrations" in Chapter 3, "Migrating NAS 2.1 Applications."

- **Complete migration to the new programming model.** This approach requires a lot of development resources and involves a full redesign, but it enables the application to take full advantage of the features of the new programming model.

  This approach is described in Chapter 3, "Migrating NAS 2.1 Applications."

Migrating NAS 4.0 Applications to iPlanet Application Server 6.0

NAS 4.0 uses Netscape and older Java standards which have been replaced with J2EE 1.2 standards in iPlanet Application Server 6.0. You need to replace deprecated methods and redeploy your applications with the new XML descriptors. Tools are provided to help with the process. For more information, see Chapter 4, "Running NAS 4.0 Applications."

Redeploying iPlanet Application Server 6.0 SP1 Applications

This section describes how you can use the new redeploy command utility to redeploying your iPlanet Application Server 6.0 SP1 applications to iPlanet Application Server 6.0 SP3.

Note that you are not required to redeploy existing applications from iPlanet Application Server 6.0 SP2 to SP3.

This section describes the following topics:

- About the Redeploy Command Utility

- Using the Redeploy Command Utility

- Limitations Using the Redeploy Command Utility

- Running the Redeploy Command Utility

About the Redeploy Command Utility

If you are upgrading an iPlanet Application Server 6.0 SP1 installation to iPlanet Application Server 6.0 EE SP3, you would see the following message at the end of the installation procedure:

Windows Platforms:

```
You have upgraded from iAS 6.0 SP1 to iAS 6.0 SP3. You must
re-deploy applications previously deployed on SP1 using the
utility <install-dir>/ias/bin/redeploy.exe.
```

Solaris Platforms:

```
You have upgraded from iAS 6.0 SP1 to iAS 6.0 SP3. You must
re-deploy applications previously deployed on SP1 using the
utility <install-dir>/ias/bin/redeploy.
```

where *<install-dir>* refers to the root directory of iPlanet Application Server installation.

A new command `redeploy` utility is provided to complete the deployment of existing applications after the upgrade to SP3. This is due to the following changes in iPlanet Application Server 6.0 SP3:

- iPlanet Application Server 6.0 directly extracts class files in a .war file under `<install-dir>`/ias/APPS directory. From iPlanet Application Server 6.0 SP2, the class files are extracted under the directory `<install-dir>`/ias/APPS/<app-name>/<module-name>/classes. The JASPER compiler integrated with iPlanet Application Server 6.0 SP3 specifically expects this directory structure. For more information, refer to iPlanet Application Server 6.0 SP3 *Release Notes*, under the Jasper Integration section in Core Server Enhancements within the "What's New in iPlanet Application Server 6.0 SP3" section.

  The Fortune application which ships with iPlanet Application Server 6.0 (all versions). In iPlanet Application Server 6.0 SP1, on deploying this application, the class file `FortuneServlet`, which constitutes this application, is extracted directly under `<install-dir>`/ias/APPS/sample/fortune, according to its package structure: sample.fortune.FortuneServlet.

  In iPlanet Application Server 6.0 SP2 and SP3, a directory called `<install-dir>`/ias/APPS/fortune/fortune is created.Within this directory, a directory called WEB-INF is created. This WEB-INF directory contains a classes directory containing sample/fortune/FortuneServlet.class and the web.xml file.

  The following information shows the difference in the locations where the class sample.fortune.FortuneServlet is extracted after deploying the Fortune Application on iPlanet Application Server SP1and SP3:

  Fully Qualified ClassName

  sample.fortune.FortuneServlet

  Location where .class file is extracted in SP1

  `<install-dir>`/ias/APPS/samples/fortune

  Location where .class file is extracted in SP3

  `<install-dir>`/ias/APPS/fortune/fortune/WEB-INF/classes/sample s/fortune

- The new iASClassLoader implemented in iPlanet Application Server 6.0 SP3 can not load classes if class files from previously deployed .war components of .ear files are placed directly under the `<install-dir>`/ias/APPS directory.

  This is because the `<install-dir>`/ias/APPS directory is in the kjs classpath. After deploying applications on iPlanet Application Server 6.0 SP1, the servlet classes can be found directly under the directory `<install-dir>`/ias/APPS. Any class which is in the kjs classpath is loaded by the system class-loader, while the ejb classes which are under a different directory are loaded by the iPlanet Application Server classloader. However, if a servlet class is found directly under the kjs classpath, the system

classloader loads it. Note that the ejb classes are not in the kjs classpath and require the iPlanet Application Server classloader to load them. Once the system classloader is invoked, the responsibility for loading the ejb classes cannot be delegated to the iPlanet Application Server class loader.

- Therefore an application consisting of a servlet invoking an ejb method gives `ClassDefNotFoundException`. For more information, refer to iPlanet Application Server 6.0 SP3 *Release Notes*, under the Dynamic Class Reloading section in Core Server Enhancements within the "What's New in iPlanet Application Server 6.0 SP3" section.

  In addition, the `redeploy` utility described later in this section moves certain class files for applications deployed on iPlanet Application Server 6.0 SP1 to a different directory, to ensure that after redeploying, both the servlet classes and the ejb classes invoked by these servlet classes are loaded by the iPlanet Application Server ClassLoader.

- User authentication has been modified in iPlanet Application Server 6.0 SP3. In particular, if an application tries to read username and password from an HTTP request string, it would not work in iPlanet Application Server 6.0 SP3. The user name and password are currently removed from the HTTP request by the Form Authentication module of iPlanet Application Server. For more information, see "Limitations" later in this section.

Using the Redeploy Command Utility

The redeploy command utility provided with iPlanet Application Server 6.0 SP3 enables you to redeploy applications which were deployed on iPlanet Application Server 6.0 SP1 before upgrade to iPlanet Application Server 6.0 SP3.

This utility is present in the installation as:

   `<install-dir>`/ias/bin/redeploy (shell script on Solaris platforms)

   `<install-dir>`/ias/bin/redeploy.exe (binary on Windows platforms)

where `<install-dir>` refers to the root directory of iPlanet Application Server installation.

You can run this utility once after the upgrade from iPlanet Application Server 6.0 SP1 to SP3. Note that you are not required to redeploy existing applications from iPlanet Application Server 6.0 SP2 to SP3.

The `redeploy` utility finds all applications already deployed on iPlanet Application Server 6.0 SP1, and their corresponding .ear files and redeploys each such application on iPlanet Application Server 6.0 SP3.

The `redeploy` utility also moves class files from .war components of applications previously deployed on iPlanet Application Server 6.0 SP1 to a trash directory. This is done so that the new iPlanet Application Server class loader can function correctly.

Note that your class files extracted from iPlanet Application Server 6.0 SP1 installation are not deleted. They are moved to the `<install-dir>`/ias/trash/sp1 directory, and placed in a directory corresponding to the package structure of that class. For example, on deploying the `Fortune` application, you obtain the `FortuneServlet` class under `<install-dir>`/ias/APPS/sample/fortune directory. The `redeploy` utility would move this file to the `<install-dir>`/ias/trash/sp1/sample/fortune directory.

Limitations Using the Redeploy Command Utility

This section describes the limitations regarding how you use the redeploy command utility.

- The `redeploy` utility will not find out applications deployed on iPlanet Application Server 6.0 SP1 which consist of only .war files (not packaged within any .ear file). Such applications have to be re-deployed manually by the user after the upgrade.

- If an application deployed on iPlanet Application Server 6.0 SP1 uses a database client not supported on iPlanet Application Server 6.0 SP3, the `redeploy` utility cannot manage such situations. You must register a database client compatible with iPlanet Application Server 6.0 SP3 using the `db_setup` utility, and register the datasources for these applications.

- Your application must not contain code which tries to read username and password from a HTTP request. When running such an application, an Authentication Error will result even after re-deploying the application on iPlanet Application Server 6.0 SP3.

Example

Consider the following code snippet from the `BankLoginServlet.java` file which was shipped with iPlanet Application Server 6.0 SP1:

```
public class BankLoginServlet extends HttpServlet {

public void doGet (HttpServletRequest request,HttpServletResponse
response)

    throws ServletException, IOException {
        response.setContentType("text/html");

        PageDispatcher dispatcher = new PageDispatcher(this,
        request,response);

        PrintWriter out = response.getWriter();
```

```
    String userName = request.getParameter(Constants.USER_NAME);
    String passWord = request.getParameter(Constants.PASSWORD);

    if ((userName == null || userName.equals("")) || (passWord ==
    null || passWord.equals("")))

    {
    request.setAttribute("error", new ErrorReporter("Login
    failed: username or password was null"));

    dispatcher.showPage(Constants.LOGIN_ERROR_PAGE);

    return;
    }

  }

}
```

The above code tries to parse the `HttpRequest` string for the values of `username` and `password`. However, retaining the `username` and `password` in the `HttpRequest` string is a security loophole. This has been corrected in iPlanet Application Server 6.0 SP3, that is, the `username` and `password` are removed from the `HttpRequest` string after Form-based authentication in SP3. Thereafter the user program would not be able to read the values of username and password from the `HttpRequest`. Applications having code as shown above, would not work on iPlanet Application Server 6.0 SP3 even after re-deployment.

Running the Redeploy Command Utility

To run the `redeploy` utility, perform the following steps:

1. Please ensure that you run the redeploy utility once after upgrading from SP1 to SP3.

    On Solaris platforms:

    ```
    cd <install-dir>/ias/bin

    ./redeploy
    ```

    On Windows platforms:

    ```
    cd <install-dir>/ias/bin redeploy
    ```

2. Restart the iPlanet Application Server before running any of the re-deployed applications.

Running NAS 2.1 Applications

This chapter describes how to run NAS 2.1 applications on iPlanet Application Server 6.0 without making any source-level changes. Information on setting up the online Bank example is at the end of the chapter. It is recommended you set up the Bank example first before running your own applications on iPlanet Application Server 6.0. The Bank example will help you walk through a step by step migration.

iPlanet Application Server 6.0 is completely backward-compatible with NAS 2.1. In other words, you should be able to deploy your older NAS 2.1 application on iPlanet Application Server 6.0 without code alteration. However, implementation requires some steps. For example, C++ applications and extensions must be recompiled before deploying on the new server (see "C++ Applications and Extensions" on page 28). Also, you must have correct the class path for the version of JDBC you use.

This chapter includes the following sections:

- NAS 2.1 Application Components

- Deploying NAS 2.1 Applications

- Java Extensions

- C++ Applications and Extensions

- Beginning the Migration Process

- Migrating the Sample Applicatons

NAS 2.1 Application Components

This section describes the iPlanet Application Server 6.0 support for each of the major types of components from the 2.1 programming model. This support is outlined in the following sections:

- HTML Templates
- AppLogics
- Database Logic: DAE and JDBC
- NAS Registry

HTML Templates

For presentation layout, NAS 2.1-style HTML Templates, including GX tags, are fully supported without alteration by the NAS template engine. If a template is called by a servlet, however, it is compiled as a JSP. JSPs support GX tags with the exception of hierarchical queries.

AppLogics

The AppLogic framework is fully supported in iPlanet Application Server 6.0, though many of the proprietary APIs introduced in NAS 2.1 have been deprecated in favor of the Java standards on which the new programming model is based. For more information, see *iPlanet Application Server Foundation Class Reference*.

Database Logic: DAE and JDBC

The NAS 2.1 database access classes and interfaces are now deprecated in favor of JDBC, the Java standard database connectivity API. Code that uses NAS 2.1 database connection and query methods is supported in iPlanet Application Server 6.0, but this support may disappear in a future release.

The new JDBC layer provides the same functionality as the old 2.1 JDBC layer and many new methods are supported. As a result, you may want to modify some of your Applogic code to remove workarounds or add new JDBC calls.

AppLogics should use either of the foillowing JDBC layers:

- The new JDBC layer.

- The old JDBC layer, but not both. Servlets and EJBs should use only the new JDBC layer. Mixing and matching of JDBC calls from each version is not supported.

You can use JDBC AppLogics from NAS 2.1 against the same iPlanet Application Server 6.0 JDBC layer, but you must make sure that the JDBC 2.0 interfaces are loaded into the JVM instead of the 1.2 interfaces. For example, if you get a log message like the following, you probably have the JDBC 1.2 interfaces in your CLASSPATH before the JDBC 2.0 interfaces:

```
[01/05/99 11:25:51:0] error: APPLOGIC-caught_exception: Caught
Exception:
java.lang.NoSuchMethodError: java.sql.Statement: method
addBatch(Ljava/lang/String;)V not found
```

NAS Registry

Part of the NAS registry now resides in an LDAP directory, though for the most part access to it has not changed. For more information, see the *Administration & Deployment Guide*.

Deploying NAS 2.1 Applications

Use the iPlanet Application Server Administrator Tool to deploy all applications to iPlanet Application Server. For more information, see the *Administration & Deployment Guide*.

Java Extensions

To migrate Java NAS 2.1 extensions to iPlanet Application Server 6.0, perform the following steps:

1. Load IDL code in iPlanet Extension Builder 6.0 and create new generated code.

2. Merge any changes which have been made to the previous extensions into the new generated code.

3. Convert any references to NMI to JNI (if applicable).

4. Perform all other Java code changes for JDK 1.2.2.

5. Recompile all code.

C++ Applications and Extensions

iPlanet Application Server 6.0 provides new versions of required C++ header files. For this reason, C++ applications and extensions must be recompiled using the new header files. Users must recompile and link their extensions against iPlanet Application Server 6.0 libraries.

iPlanet Application Server provides pre-built extensions for several legacy systems, including MQSeries, TUXEDO, and CICS. These extensions have been re-released to provide support for iPlanet Application Server 6.0, though they will retain support for the NAS 2.1 programming model.

Beginning the Migration Process

When you decide to migrate your application to the new model, it is easiest to begin by redesigning your application and coming up with a transition plan. It is often best to gradually migrate parts of an application to a new programming model, rather than planning a large-scale migration for the entire application. However, it is also possible to migrate gradually by programming tier (presentation layout, business logic, and so on) rather than by application component.

For information on partial migration, including how to allow 2.1 components (like AppLogics) to interact with 6.0 components (like servlets and EJBs), see "Partial Component Migrations" in Chapter 3, "Migrating NAS 2.1 Applications."

Migrating the Sample Applicatons

This section describes how you can migrate the following example applications:

- Online Bank Example

- US Population Java Sample Application

- US Population C++ Sample Application

Online Bank Example

This example is divided into the following sections:

- Migration From NAS 2.1 to iPlanet Application Server 6.0 - Solaris Only

- C++ COnlineBank(NAS2.1) Sample Application

Migration From NAS 2.1 to iPlanet Application Server 6.0 - Solaris Only

iPlanet Application Server 6.0 includes JDK 1.2.2 as part of its distribution. With the new JDK1.2 all the java core packages like `java.io`, `java.lang`, and so on, are packaged into `rt.jar` instead of the classes.zip as in JDK 1.1. You can find `rt.jar` in the following location:

/iAS6.0-install-directory/nas/usr/java/jre/lib

If you want to compile any java classes you have to have `rt.jar` in the `CLASSPATH` and you should use the `javac`, located in the /iAS6.0-install-directory/nas/usr/java/bin directory, for compilation.

To migrate from NAS 2.1 to iPlanet Application Server 6.0 (Solaris platforms), perform the following steps:

1. Copy the OnlineBank java Application package to the machine where iPlanet Application Server 6.0 has been successfully installed by creating a new directory OnlineBank under `GXAPP`.

    /iAS6.0-install-directory/nas/APPS/GXApp/OnlineBank

2. Create a new directory called OnlineBank under docs/GXApp in the webserver installation directory.

    /Netscape/Suitespot/docs/GXApp/OnlineBank

    Then copy all the HTML files of OnlineBank in NAB 2.1 webserver docs directory into the OnlineBank directory.

3. Create an entry for `ksample` in the `tnsnames.ora`. Make sure `ksample` points to the Oracle database running 8.0.5.

4. Register using `kreg` utility.

    Change directory to /iAS6.0-install-directory/nas/APPS/GXApp/OnlineBank then run /iAS6.0-install-directory/nas/bin/kreg `OnlineBank.gxr`.

    Make sure that you are running the iPlanet Application Server 6.0 kreg and not the old 2.1 version.

5. Run the sample application http://hostname/GXApp/OnlineBank/OBLogin.html, and then go further to see the account balance and other options.

C++ COnlineBank(NAS2.1) Sample Application

To run the C++ COnlineBank(NAS2.1) Sample Application, perform the following steps:

1. Copy the COnlineBank Application to the machine where iPlanet Application Server 6.0 has been successfully installed by creating a new directory COnlineBank under GXApp /iAS6.0-install-directory/nas/APPS/GXApp/COnlineBank,

2. Create a new directory called COnlineBank under docs/GXApp in the webserver installation directory: /Netscape/Suitespot/docs/GXApp/COnlineBank, and copy all the HTML files of COnlineBank in NAB 2.1 webserver docs directory into the above created COnlineBank directory.

3. Set the following two environment variables:

   setenv GX_ROOTDIR /iAS6.0-install-directory/nas

   setenv GX_ROOT /iAS6.0-install-directory/nas

4. Create an entry for `ksample` in the `tnsnames.ora`. Make sure `ksample` points to the Oracle Database running 8.0.5.

5. Run make with the supplied makefile: /usr/ccs/bin/make -f makefile (Where /usr/ccs/bin is the directory where the make file exists). This action copies the generated libCOnlineBank.so file into the /iAS6.0-install-directory/nas/gxlib directory.

6. Register using `kreg` utility.

   Change the directory to /iAS6.0-install-directory/nas/APPS/GXApp/COnlineBank then run /iAS6.0-install-directory/nas/bin/kreg `COnlineBank.gxr`.

   Make sure that you are running the new iPlanet Application Server 6.0 kreg and not the old NAS 2.1 version.

7. Run the sample application http://hostname/GXApp/COnlineBank/COBLogin.html and check the account balance and other options.

US Population Java Sample Application

To migrate the US Population Java example from NAS 2.1 to iPlanet Application Server 6.0, perform the following steps:

1. Copy the US Population java application package to a new directory called States on the machine where iPlanet Application Server 6.0 has been successfully installed (for example: GXAPP/IAS6.0-install-directory/nas/APPS/GXApp/States).

2. Create a new directory called States under docs/GXApp in the webserver installation directory:

   /Netscape/Suitespot/docs/GXApp/States

   Then, copy all the HTML files from the US Population application in NAB 2.1 webserver docs directory into the above named States directory.

3. Create an entry for `ksample` in the `tnsnames.ora`. Make sure `ksample` points to the Oracle DataBase version 8.0.5.

4. Register using kreg utility.

   Then change the directory to /IAS6.0-install-directory/nas/APPS/GXApp/States, and then run /IAS6.0-install-directory/nas/bin/kreg `states.gxr`.

   Make sure that you are running theiPlanet Application Server 6.0 `kreg` and not the old 2.1 version.

5. Run the US Population application:

   http://hostname/GXApp/States/index.html

   Then click on `RunRegionReport` to see the population statistics.

US Population C++ Sample Application

To migrate the US Population C++ example from NAS 2.1 to iPlanet Application Server 6.0, perform the following steps:

1. Copy the US Population C++ application to the machine where iPlanet Application Server 6.0 has been successfully installed. Copy it to a new directory called CStates under GXAPP (for example: /IAS6.0-install-directory/nas/APPS/GXApp/CStates).

2. Create a new directory called CStates under docs/GXApp in the webserver installation directory:

   /Netscape/Suitespot/docs/GXApp/CStates.

   Then copy all the HTML files of US Population in NAB 2.1 webserver documents directory into the above created CStates directory.

3. Create an entry for `ksample` in the `tnsnames.ora`. Make sure `ksample` points to the Oracle DataBase version 8.0.5.

4. Run `nmake` on the states.mak file (nmake -f states.mak).

5. Copy the generated states.dll to /IAS6.0-install-directory/nas/bin directory.

6. Register using `kreg` utility.

   Then change directory to /IAS6.0-install-directory/nas/APPS/GXApp/CStates, and run /IAS6.0-install-directory/nas/bin/kreg `states.gxr`.

   Make sure that you are running the iPlanet Application Server 6.0 `kreg` and not the older 2.1 version.

7. Run the US Population application:

   http://hostname/GXApp/CStates/index.html

   Then click on `RunRegionReport` to see the population statistics.

Migrating NAS 2.1 Applications

This chapter describes altering your NAS 2.1 applications to fit the iPlanet Application Server 6.0 programming model.

This chapter includes the following sections:

- Redesigning Your Application

- Migrating Presentation Logic

- Migrating Business Logic

- Migrating Data Access Logic

- Partial Component Migrations

Redesigning Your Application

When redesigning an existing application, it is important to keep in mind that changes made to one part will affect the others.

It may be useful to think of your application as one of the following models:

- A series of user interactions to reach a goal. Example: an online survey or standardized test.

- An activity clearinghouse with a central front page. Example: an online bank, with a central page that leads to several activities (that is, withdrawals, transfers, and so on).

In reality, your application is likely to be a combination of the two. For example, an online bank could really be a central clearinghouse where each of the pathways leads to a series of user interactions to reach a goal.

However your application is subdivided, it is often best to migrate one part at a time. For more details, see "Partial Component Migrations" on page 37.

Migrating Presentation Logic

This section describes the following concepts:

- Recreating AppLogics as Servlets

- Recreating Presentation Layout

- Recreating Sessions and Security

Recreating AppLogics as Servlets

AppLogics map directly to servlets. They are similar in that they are both called by URLs, and they both contain mechanisms to process input and generate output. The main difference, besides the layout of the code itself, is that servlets generally do not perform business logic, as AppLogics do. Rather, business logic is handled in EJBs and referenced by the servlet, similarly to the way presentation layout is handled in JSPs and referenced by the servlet. In short, a servlet is like an AppLogic with the business logic re-implemented in a separate entity.

For information about servlets, see Chapter 3, "Controlling Applications with Servlets," in the *Developer's Guide (Java)*.

Servlets must contain a `service()` method (or, for HTTP servlets, this can be implemented as `doGet()`, `doPost()`, etc. depending on the HTTP transport method), which is logically similar to the `execute()` method in an AppLogic. This is the main flow of execution for the component.

Moreover, where iPlanet Application Server creates an `IValList` member variable to contain incoming data for an AppLogic, for servlets, iPlanet Application Server instead creates a request object and passes it as a parameter to the servlet. Likewise, where AppLogics use an `IValList` for output, servlets use a response object, also passed to the servlet as a parameter. The following code examples illustrate both cases:

AppLogic

```
public class MyAppLogic extends AppLogic {
    public void execute () throws IOException {
    ...
    String lastName = valIn.getValString("lastName");
```

```
    ...
    return result ("<html><body>\n"
                + "<p>Your last name is " + lastName + ".\n"
                + "</body></html>\n");
    }
}
```

Servlet

```
public class myServlet extends HttpServlet {
public void service (HttpServletRequest req,
                     HttpServletResponse res)
          throws IOException, ServletException
    {
    ...
    res.setContentType("text/html");
    String lastName = req.getParameter("lastName");
    ...
    PrintWriter output = res.getWriter();
    output.println("<html><body>\n");
                + "<p>Your last name is " + lastName + ".\n"
                + "</body></html>\n");
    }
}
```

Note that you can also reimplement an AppLogic as a JSP, since JSPs and servlets are more or less the same entity from different viewpoints. For example:

```
<html><body>
<p>Your last name is <display property="request:params:lastName">.
</body></html>
```

For information about servlets, see Chapter 4, "Presenting Application Pages with JavaServer Pages," in the *Developer's Guide (Java)*.

Recreating Presentation Layout

In a sense, your 2.1 HTML templates are already migrated. The iPlanet Application Server 6.0 template engine simply compiles these templates as if they were JSPs. The new template engine supports GX tags for backward compatibility, with the exception of hierarchical queries.

However, GX tag support in JSPs is deprecated, so these templates must be converted to use standard JSP tags and syntax. JSPs use beans to encapsulate output parameters, and can access arbitrary Java objects as well. You can even access EJBs directly from JSPs. Normally, however, you set attributes in the request object during the execution of a servlet and then recall them in a JSP.

For more details about JSPs, including examples, see Chapter 4, "Presenting Application Pages with JavaServer Pages," in the *Developer's Guide (Java)*.

Recreating Sessions and Security

iPlanet Application Server 6.0 sessions use the `HttpSession` interface. The concepts are similar to the way sessions worked in NAS 2.1, though the API is different. A servlet (or AppLogic) creates a session, thereby instantiating a session object that persists for the life of the user session. A session cookie is returned to the client and reread on subsequent interactions with that client. Once the session exists, you can bind objects to it.

Security in servlets has changed. For more information, see "Understanding the Security Model," in Chapter 12, "Writing Secure Applications," in the *Developer's Guide (Java)*.

Migrating Business Logic

Business logic is handled in iPlanet Application Server 6.0 through Enterprise JavaBeans (EJBs) rather than in AppLogics. An important distinction between AppLogics and EJBs is that EJBs can be made to be persistent during a "session" with the user, separately designated from the user's session, in the case of session beans. Entity beans exist independently of users, and thus potentially persist through the life of the server.

You write these EJBs to perform discrete tasks, then connect to them from servlets. For example you would do this if you have an electronic shopping cart.

For details on JDBC and transaction support, see Chapter 8, "Handling Transactions with EJBs," and Chapter 8, "Handling Transactions with EJBs," in the *Developer's Guide (Java)*.

Migrating Data Access Logic

This section describes redeploying database calls using the JDBC API.

The JDBC layer in iPlanet Application Server 6.0 supports 100% of the JDBC 2.0 specification and standard extensions.

For details on JDBC and transaction support, see Chapter 8, "Handling Transactions with EJBs," and Chapter 9, "Using JDBC for Database Access," in the *Developer's Guide (Java)*.

The JDBC 2.0 interfaces provided in `$GX_ROOTDIR/solarisdbg/JDK_1.2/java` (or similar directory) must be before any other JDBC interfaces in the `CLASSPATH`. iPlanet Application Server 6.0 works with JDK 1.2 which has JDBC 2.0 interfaces in `$JAVA_HOME/lib/rt.jar`, so make sure this `rt.jar` is after the iPlanet Application Server provided classes, as follows:

```
setenv CLASSPATH
:$GX_ROOTDIR/solarisdbg/JDK_1.1:...:$JAVA_HOME/lib/rt.jar:
...
```

Incompatibility Errors

If you get a log message like the following, you probably have the JDBC 1.2 interfaces in your CLASSPATH before the JDBC 2.0 interfaces:

```
[01/05/99 11:25:51:0] error: APPLOGIC-caught_exception:
Caught Exception:
java.lang.NoSuchMethodError: java.sql.Statement: method
addBatch(Ljava/lang/String;)V not found
```

Partial Component Migrations

This section describes how to use older components (Java and C++ AppLogics) with newer components (servlets and EJBs). The following four combinations are supported:

- Calling EJBs from Java AppLogics

- Calling Servlets from Java AppLogics

- Calling Java AppLogics from Servlets

- Calling C++ AppLogics from Servlets

Calling EJBs from Java AppLogics

Since there is no special context shared between servlets and EJBs, you call an EJB from an AppLogic in exactly the same way you would from a servlet.

This example shows an AppLogic accessing an EJB called ShoppingCart. The AppLogic creates a handle to the cart by casting the user's session ID as a ShoppingCart after importing the cart's remote interface. The cart is stored in the user's session.

```
import cart.ShoppingCart;
// Get the user's session and shopping cart
    //first create the session
    ISession2 sess = createSession(GXSESSION.GXSESSION_DISTRIB,
                                    0,              //no timeout
                                    "callEjb",    //app name
                                    null,          //system-gen'd ID
                                    null);

    //create an IValList to store the shopping cart in the session
    IValList ival = sess.getSessionData();

    ShoppingCart cart = (ShoppingCart)ival.getVal("shoppingCart");

    // If the user has no cart, create a new one
    if (cart == null) {
        cart = new ShoppingCart();
        ival.setVal("shoppingCart", cart);
    }
```

You can access EJBs by using the Java Naming Directory Interface (JNDI) to establish a handle, or proxy, to the EJB. You can then refer to the EJB as a regular object; any overhead is managed by the bean's container.

This example shows the use of JNDI to look up a proxy for a shopping cart:

```
String jndiNm = "Bookstore/cart/ShoppingCart";
javax.naming.Context initCtx;
Object home;
    try {
            initCtx = new javax.naming.InitialContext(env);
    } catch (Exception ex) {
            return null;
    }
    try {
            java.util.Properties props = null;
            home = initCtx.lookup(jndiNm);
    }
    catch(javax.naming.NameNotFoundException e)
    {
            return null;
    }
    catch(javax.naming.NamingException e)
    {
```

```
        return null;
    }
    try {
        IShoppingCart cart = ((IShoppingCartHome) home).create();
        ...
} catch (...) {...}
```

Calling Servlets from Java AppLogics

You can call a servlet from a Java AppLogic, for example if you want your AppLogic to call a JSP, using GXContext.NewRequest() or GXContext.NewRequestAsync(). For more details and specific examples of NewRequest(), see the documentation for the GXContext class, in the *iPlanet Application Server Foundation Reference.*

You can also call a JSP from an AppLogic, since JSPs and servlets are the same type of object after instantiation.

To call a servlet from an AppLogic using the same process call the servlet's serlvet engine (an AppLogic called ServletRunner), consider the following example:

```
class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","nsOnlineBank");
        valIn.setValString("servletName","Login");
        valIn.setValString("SCRIPT_NAME","nsOnlineBank/Login");
        com.netscape.server.servlet.servletrunner.ServletRunner sr =
            new
com.netscape.server.servlet.servletrunner.ServletRunner();
        sr.valIn = valIn;
        sr.valOut = valOut;
        sr.context = context;
        sr.stream = this.stream;
        sr.ticket = this.ticket;
        sr.request = this.request;
        sr.COMSet(COMGet());
        sr.COMAddRef();
        sr.execute();
        ...
    }
}
```

To call a servlet from an AppLogic in a new process using NewRequest(), consider the following example:

```
class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","nsFortune");
        valIn.setValString("servletName","fortune");
        valIn.setValString("SCRIPT_NAME","nsOnlineBank/Login");
        retValue = GXContext.NewRequest(m_Context,
                                    "ApplogicServlet_nsFortune_fortu
ne",
                                    valIn,valOut,host,port,0);
        ...
    }
}
```

You can call a JSP in much the same way, as in the following example:

```
public class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","System");
        valIn.setValString("servletName","JSPRunner");
        valIn.setValString("JSP","nsOnlineBank/jsp/abc.jsp");
        valIn.setValString("SCRIPT_NAME","nsOnlineBank/Login");
        retValue =
            GXContext.NewRequest(m_Context,
                                "Applogic
Servlet_System_JSPrunner",
                                valIn,valOut,host,port,0);
    ...
    }
}
```

To call a servlet using a GUID, consider the followin example:

```
public class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","nsFortune");
        valIn.setValString("servletName","fortune");
        newRequest("{6F3547D0-FDCB-1687-B323-080020A16896}",
                    valIn,valOut,0);
    }
}
```

Calling Java AppLogics from Servlets

You can call AppLogics from servlets using GXContext.NewRequest() or
GXContext.NewRequestAsync(). For more details and specific examples, see the
documentation for the GXContext class, in the *iPlanet Application Server Foundation
Class Reference*.

In order to call an AppLogic using `NewRequest()`, you must first cast the server's context to an `IContext` object, and then set up the input and output `IValList` objects for the AppLogic.

This example shows how to obtain an `IContext` object, set up parameters for the AppLogic, and finally call the AppLogic using `NewRequest()`:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.kivasoft.applogic.*;
import com.kivasoft.types.*;
import com.netscape.server.servlet.extension.*;

public class callAnAppLogic extends HttpServlet {

    public void service(HttpServletRequest req,
                        HttpServletResponse res)
        throws ServletException, IOException
    {
        // first set up ic as a handle to an IContext
        ServletContext sctx = getServletContext();
        com.netscape.server.IServerContext isc;
        isc = (com.netscape.server.IServerContext) sctx;
        com.kivasoft.IContext ic = isc.getContext();

        //set up IValLists and GUID
        IValList vi = GX.CreateValList(); // valIn
        valIn.setValString("randomParameter", "Cirdan the
Shipwright");

        IValList vo = GX.CreateValList(); // valOut

        String al = req.getParameter("AppLogicToCall");
                    // expect AppLogicToCall in request

        //finally, call the AppLogic
        GXContext.NewRequest(ic, al, vi, vo, 0);
    }
}
```

Accessing the Servlet's AppLogic

Each servlet is contained in an AppLogic. You can access the AppLogic instance controlling your servlet using the method `getAppLogic()` in the iPlanet Application Server feature interface `HttpServletRequest2`.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.kivasoft.applogic.*;
import com.kivasoft.types.*;
import com.netscape.server.servlet.extension.*;7

public class callAnAppLogic extends HttpServlet {

    public void service(HttpServletRequest req,
                         HttpServletResponse res)
        throws ServletException, IOException
    {
        HttpServletRequest2 req2 = (HttpServletRequest2)req;
        AppLogic al = req2.getAppLogic();
        //al is now a handle to the superclass
        ...
    }
}
```

Calling C++ AppLogics from Servlets

The method GXContext.NewRequest() as described in "Calling Servlets from Java AppLogics" on page 39, calls an AppLogic by GUID and provides handles to objects as input and output parameters. This method works for calling C++ AppLogics as well as Java AppLogics, since the AppLogic is called by the specified name or GUID and not by a handle specific to Java. See the example shown in that section.

Sessions in Partially Migrated Applications

The HttpSession2 interface is an additional session interface that gives you direct access to the session object. Using this interface, you can share sessions (and therefore data) between applogics and servlets.

In servlets, a session is an instance of HttpSession. But in AppLogics, session data is an IValList object. An AppLogic stores integers, strings, and blobs (byte arrays) in a session, whereas a servlet stores serializable objects in a session. As a result, there is no immediate mapping between what an AppLogic stores and what a servlet stores in a session (except for strings).

The `HttpSession2` interface solves the issue of sharing session data. `HttpSession2` provides methods for storing and retrieving integers, strings, blobs, and user login data—methods that parallel what an AppLogic developer uses. In this way, `HttpSession2` enables sessions to work back and forth across AppLogics and servlets.

`HttpSession2` provides `loginSession( )` and `logoutSession( )` for servlets to share the AppLogic session API. These methods have been deprecated in iPlanet Application Server 6.0. These two methods are typically used with `isAuthorized( )`, as is done for AppLogics. Servlets are also registered with an access control list, so that a secure session established in an AppLogic can be used in a servlet, and vice versa.

For more information, see Chapter 12, "Writing Secure Applications," in the *Developer's Guide (Java)*.

Making the Session Visible

Note that, because sessions are controlled with cookies, a session created in an AppLogic is not visible in a servlet by default. This is because cookies are domain- and URI-dependent, and the URI for a servlet is different from that of an AppLogic. To work around this problem, call `setSessionVisibility()` before you call `saveSession()` when you create a session in an AppLogic.

It is important to do this before calling `saveSession()`, since saving the session also creates the session cookie.

For example, in an AppLogic, consider the following example:

```
domain=".mydomain.com";
path="/"; //make entire domain visible
isSecure=true;
if ( setSessionVisiblity(domain, path, isSecure) == GXE.SUCCESS )
    { // session is now visible to entire domain }
```

For more information about sessions, see Chapter 11, "Creating and Managing User Sessions," in the *Developer's Guide (Java)*.

Converting ITemplateData to ResultSet

NAS 2.1 provided an interface called ITemplateData to represent a hierarchical source of data used for HTML template processing. In NAS 2.1, `ITemplateData` (Java) provides methods for iterating through rows in a set of memory-based hierarchical data and retrieving column values. This functionality is not supported in iPlanet Application Server 6.0, although group names are supported (and required).

In iPlanet Application Server 6.0, `ITemplateData` functionality is replaced with JDBC `ResultSet` objects. You can convert `ITemplateData` objects to `ResultSet` objects using the method `convertITemplateDataToResultSet()` from the `BaseUtils` class. For specific usage information, see the documentation for the `BaseUtils` class in the *iPlanet Application Server Foundation Class Reference*. The following example shows an `ITemplateData` conversion to a `ResultSet` in an AppLogic. Note that you must provide a data group name as a parameter to the conversion method.

```
ITemplateData itd = GX.CreateTemplateDataBasic("myTemplateData");
... // populate myTemplateData
...
ResultSet rs =
BaseUtils.convertITemplateDataToResultSet("dataGroup1",
                                                    itd);
```

Running NAS 4.0 Applications

This chapter describes the basic steps to migrate your Netscape Application Server 4.0 applications to run on iPlanet Application Server 6.0.

This chapter contains the following sections:

- Overview

- Differences Between NAS 4.0 and iPlanet Application Server 6.0

- Migrating NAS 4.0 Components

- Migration Example "The Bank"

Overview

iPlanet Application Server 6.0 is certified compliant with Java 2 Platform, Enterprise Edition specification version 1.2 (J2EE 1.2). While the architecture of iPlanet Application Server 6.0 is the same as NAS 4.0, the 100% J2EE standard means that your applications must conform to J2EE 1.2 in order to run.

The migration effort depends on how much your applications depend on deprecated J2EE and NAS proprietary methods. Deployment and JavaServer Pages require conversion procedures. Tools are provided for these procedures. In general your effort will be to replace deprecated methods, convert and redeploy. One way to check if your applications have deprecated methods is to recompile them.

Follow the steps outlined in this chapter to begin the migration of your application. You may find it helpful to work through the "Bank" migration example which is available online at the iPlanet web site. Go to http://www.iPlanet.com/support for further information. There is a step by step migration of a NAS 4.0 application to run on iPlanet Application Server 6.0 available.

Differences Between NAS 4.0 and iPlanet Application Server 6.0

The following table highlights the main differences between NAS 4.0 and iPlanet Application Server 6.0 components. Each of these component differences and the procedure to migrate, follows the table.

Component	NAS 4.0	Migration Tactic	Effort	iPlanet Application Server 6.0
JDK	JDK 1.1.7	See JDK Migration	Medium	JDK 1.2.2
Servlets	Servlet 2.1	See Servlet Migration	Low/None	Servlet 2.2
Servlet Deployment	Uses NTV Deployment descriptors	Use tool to convert to XML. See Servlet Deployment	Medium	Uses XML Descriptors
JSP	JSP 0.92	See JSP Migration	Medium	JSP 1.1
EJB	EJB 1.0	See EJB Migration		EJB 1.1
EJB Deployment	Uses Property file Deployment descriptors	Use tool to convert to XML. See EJB Deployment.	Medium	Uses XML Descriptors
JNDI	JNDI 1.1	See JNDI Migration		JNDI 1.2
Security	ACL Checking	See Security Section	Low	declarative based

Migrating NAS 4.0 Components

This section outlines the requirements for migrating various NAS 4.0 components to iPlanet Application Server 6.0. The following topics are included here:

•   Basic Migration Steps

- JDK Migration

- Servlet Migration

- Servlet Deployment

- JSP Migration

- EJB Migration

- EJB Deployment

- JNDI Migration

- Java Extensions

- C++ Extensions

- Using Rich Client (ISecurity Interface)

Basic Migration Steps

The basic steps you need to consider for migrating NAS 4.0 components to iPlanet Application Server are as follows:

1. Look for deprecated/modified methods in your code.

2. Replace deprecated methods as shown in this chapter.

3. Convert Java Server Pages with supplied tool.

4. Convert descriptors for servlets and EJBs with the supplied tool.

5. Redeploy your application using the iPlanet Application Server Deployment Tool as described in Chapter 2, of the *Administration & Deployment Guide.*

JDK Migration

iPlanet Application Server 6.0 uses the Java 2 Development Toolkit version 1.2.2 (JDK 1.2.2).

An important difference for iPlanet Application Server is that if a native interface is used such as NMI, you need to replace it with JNI.

For more information regarding the changes from JDK 1.1.7 to JDK 1.2.2 go to:
`http://java.sun.com/products/jdk/1.2/docs/relnotes/features.html`

For more information on JDK 1.2.2 go to:
`http://java.sun.com/products/jdk/1.2/docs/index.html`

JDK Migration Steps

To migrate NAS 4.0 applications from JDK 1.1.7 to JDK 1.2.2, perform the following steps:

1. Get the list of deprecated methods and their replacements at:

   `http://java.sun.com/j2ee/j2sdkee/techdocs/api/deprecated-list.ht ml`

   For specific incompatibilities, see:

   `http://java.sun.com/products/jdk/1.2/compatibility.html`

2. Replace the deprecated methods and recompile your application.

3. Redeploy your application.

Servlet Migration

iPlanet Application Server 6.0 uses version 2.2 of the Java Servlet Specification. For detailed information on the Specification go to: http://java.sun.com/products/servlet/.

To find out what is new in Java Servlet API 2.2 go to: http:/developer.java.sun.com/developer/technicalArticles/Servlets/servletapi/

Servlets from NAS 4.0 will run as is on iPlanet Application Server 6.0 if they use interfaces from version 2.1 only and do not use any deprecated classes.

The following major changes have been made to the Java Servlet specification since version 2.1:

• Java class, configured in XML

• Servlet lives in a container

• Servlet is always part of an application

• Servlets are archived in `.war` files

• Security features have been added

• The introduction of the web application concept

• The introduction of the web application archive files

• Response buffering added

- The introduction of distributable servlets

- `RequestDispatcher` can be acquired by name

- `RequestDispatcher` can be acquired using a relative path

- Internationalization improvements

- Many clarifications of distributed servlet engine semantics

- Behaviour of servlet parameter validation has changed (for more information, see the *Developer's Guide (Java)*)

Servlet API Changes

The following servlet API changes have been implemented:

- The `getLocale` method was added to the `ServletRequest` interface to aid in determining what client locale.

- `isSecure` method was added to the `ServletRequest` interface. This indicates whether or not the request was transmitted via a secure transport such as HTTPS.

- `getInitParameter` and `getInitParameterNames` method were added to the `ServletContext` interface. Initialization parameters can now be set at the application level to be shared by all servlets that are part of that application.

- The construction methods of `UnavailableException` have been replaced as existing constructor signatures. These constructors have been replaced by simpler signatures.

- The `getServletName` method was added to the `ServletConfig` interface. This allows a servlet to obtain the name by which it is known to the system, if any.

- Added the `getHeaders` method to the `HttpServletRequest` interface to allow all the headers associated with a particular name to be retrieved from the request.

- Added the `isUserInRole` and `getUserPrinciple` methods to the `HttpServletRequest` method to allow servlets to use an abstract role based authentication.

- Added the `addHeader`, `addIntHeader`, and `addDateHeader` methods to the `HttpServletResponse` interface to allow multiple headers to be created with the same header name.

- Added the `getAttribute`, `getAttributeNames`, `setAttribute`, and `removeAttribute` methods to the `HttpSession` interface to improve the naming conventions of the API. The `getValue`, `getValueNames`, `setValue`, and `removeValue` methods are deprecated.

- Added the `getContextPath` method to the `HttpServletRequest` interface so that the part of the request path associated with a web application can be obtained.

Servlet Migration Steps

There are two paths to migration:

a. Replace both NAS 4.0 deprecated methods and J2EE deprecated methods to make your application 100% J2EE compliant

b. Replace NAS 4.0 deprecated methods only. Your application will run on iPlanet Application Server 6.0 even if it uses J2EE deprecated methods. However, it is advisable to plan to migrate to J2EE 1.2, as the deprecated methods may not be available in the future.

Note that some methods have been deprecated in HTTPSession2.

To migrate your servlets from NAS 4.0 to a 100% J2EE complliant application, you will need to replace NAS 4.0 propietary methods and J2EE deprecated methods. Follow the Optional Step 1 to accomplish this replacement.

1. (Optional) Replace Servlet 2.2 deprecated methods.

   For a list of deprecated methods go to
   http://java.sun.com/products/servlet/2.2/javadoc/deprecated-list.html

2. Replace Access Control List based logic with declarative security model.

   Use the new declarative based security procedure described in the servlet specification instead of Access Control Lists (ACL). Security is implemented as part of deployment in XML files instead of at the application level.

   a. Remove the following deprecated HTTPSession2 security methods:

   ```
   boolean loginSession(String user, String paswd);
   void logoutSession();
   boolean isAuthorized(String target, String permission);
   ```

   b. Use the auth method tag in XML files to set the authentication method to either Basic, Certificate, or Form based.

   c. In the `.xml` file, use the `<security constraint>` to specify the roles that can execute the servlets. Create roles using `<role-name>` or reference a logical role by using `<role-link>` tags.

   d. Remove the ACL entries `.gxr` files.

3. Replace URI naming that uses the AppPath as the root for absolute references to JSPs or other servlets.

    In iPlanet Application Server 6.0 the application context root is the root for absolute references. A servlet would forward to another JSP in the same application as itself, in the following manner.

    ```
    RequestDispatcher rd = req.getRequestDispatcher("/sample.jsp");
    rd.forward(req,res);
    ```

Here `sample.jsp` is in the same application as the servlet that is including it. You will find the JSP under `AppPath/ApplicationName` rather than `AppPath`, which would have been the case in NAS 4.0.

Servlet Deployment

Servlet 2.2 has introduced the use of XML files to replace the deployment descriptor used in NAS 4.0. The NTV descriptor files in your NAS 4.0 application must be converted to XML files and added to the web application archive file that the Deployment Tool creates.

1. Convert NTV descriptor files to XML files

    Use the following tool to convert the NTV files to XML:

    ```
    convertNtv2Xml $path/appInfo.ntv $newpath/myApp.xml
    ```

    `$path` points to the location of the `appInfo.ntv` (which internally provides the location of the serlvet info NTV files).

    The conversion tool creates two new files, `myApp.xml`, and `ias-myApp.xml`, in `$newpath`. These files represent the J2EE and the iPlanet Application Server-specific XML respectively.

2. Add the converted NTV descriptor files to the WAR archive file. To perform this procedure, use the following steps:

    a. Create a Servlet 2.2 WAR archive which includes servlet files and the converted descriptor `.xml` files.

    b. Launch the Deployment Tool for iPlanet Application Server 6.0.

    c. On the file menu, select "Open."

    d. Navigate to the WAR archive and choose OK.

    e. Continue adding servlet files and other files to your web application.

3. Save and deploy your web application with the Deployment Tool.

JSP Migration

iPlanet Application Server 6.0 employs version 1.1 of the Java Server Pages Specification. The JSP 1.1 specification is integrated with the J2EE, particularly for security, transaction, and session state concepts. For detailed information on the Specification, go to http://java.sun.com/products/jsp/

The JSP 1.1 specification extends JSP 0.92 to JSP 1.1 by incorporating the following enhancements:

- Using Servlet 2.2 as the foundations for its semantics.

- Enabling the delivery of translated JSP pages into JSP containers.

- Providing a portable Tag Extension mechanism.

In addition, iPlanet Application Server 6.0 provides caching and load balancing for JSPs and provides custom tag extensions.

GX Tags Deprecated

GX tags have been deprecated. Migrate any NAS 4.0 JSP templates with GX tags in them. iPlanet Application Server uses JSP extension tags instead.

JSP Migration Steps

To migrate your NAS 4.0 JavaServer Pages, perform the following steps:

1. Replace URI naming that uses the `AppPath` as the root for absolute references to servlets or other JSPs.3.

2. Convert JavaServer Pages from specification 0.92 to 1.1. JavaServer Pages must be migrated. You can use the `convert2jsp11` tool to convert JSP 0.92 to JSP 1.1, outlined in the section, Converting JSP 0.92 to JSP 1.1.

Converting JSP 0.92 to JSP 1.1

Use the supplied conversion tool (`convert2jsp11`) to convert already existing JSP 0.92 files. The tool can be used to convert individual files, or it can recurse through an entire tree of directories, converting all JSP files found.

NOTE: Remember to create back up copies of your files prior to conversion.

The conversion tool converts all the 0.92 JSP files to their 1.1 equivalent, keeping the same name. The 0.92 versions of the files are copied to a file of the same name, with the extension .0.92. For example, if you convert a file myApp.jsp, that file becomes the new JSP 1.1 version, and the older version is copied to a file called myApp.jsp.0.92.

If any of the 0.92 JSP files in a given conversion contain an error, then the conversion for that file fails, creating an empty output file. In this event, re-copy the corresponding `filename.0.92` version back to `filename`, correct the error, and run the conversion script again for that file.

Usage:

```
convert2jsp11 [-r] -ap appPath file/directory
```

Argument	Description
`-r`	Optional. Recursively convert all the JSPs in the specified directory/folder and all subdirectories/sub-folders. If this option is not given, only convert the specified file.
`-ap appPath`	Specify the `appPath` for your NAS installation (for example, `/export/nas4/nas/APPS` on Solaris, or `C:\Netscape\Server4\NAS\APPS` on Windows NT).
`file/directory`	Specify a file to convert, or a directory (with `-r`) where all files should be converted. This directory must be relative to the `appPath` given with the `-ap` option.

Examples

Note that in these examples, `appPath` is `C:\Netscape\Server4\NAS\APPS` on Windows NT and `/export/nas4/nas/APPS` on Solaris.

The following first example converts all of the JSP files in a directory called `myApplication` which is rooted in your `appPath`:

Windows NT:
```
convert2jsp11 -r -ap
c:\netscape\server4\nas\APPS myApplication
```

Solaris:
```
convert2jsp11 -r -ap /export/nas4/nas/APPS
myApplication
```

The second example, converts a single JSP file called `myJSP.jsp` in a directory called `myApplication` which is rooted in your *appPath:*

Windows NT:	`convert2jsp11 -ap c:\netscape\server4\nas\APPS` `myApplication\myJSP.jsp`
Solaris:	`convert2jsp11 -ap /export/nas4/nas/APPS` `myApplication/myJSP.jsp`

EJB Migration

iPlanet Application Server 6.0 employs version 1.1 of the Enterprise Java Bean (EJB) Specification. For detailed information on the EJB specification go to:

http://java.sun.com/products/ejb/

The EJB specification includes the following primary changes from EJB 1.0 to EJB 1.1:

• The Entity bean specification has been tightened, and support for entity beans is mandatory for Container providers. The modifications affect mainly support for transactions, Enterprise Bean Environments, Security and Deployment Descriptors. There is very little impact for EJB 1.0 applications in runtime. The only change to the runtime API of the EJB Container is the replacement of the `java.security.Identity` class with the `java.security.Principal` interface.

The following changes in the EJB 1.1 specification were made to improve the support for the development, application assembly, and deployment of iPlanet Application Server 6.0 applications.

• Support is enhanced for the enterprise bean's environment. The Bean Provider must specify all the bean's environmental dependencies using entries in a JNDI naming context.

• Support for Application Assembly in the deployment descriptor.

• Bean Provider and Application Assembler responsibilities have been clearly divided.

EJB Migration Steps

To migrate your NAS 4.0 Enterprise Java Beans, perform the following steps:

1. Replace Access Control List based logic with declarative security model.

   As with servlets, Access Control Lists on EJBs must be removed. Use the
   `<method-permission>` tag to specify the rules that can execute EJB methods.

2. Convert deployment descriptors.

   For more information, see EJB Deployment.

3. Modify code and descriptors to exclude deprecated classes and replace with new
   methods.

EJB 1.0 enterprise bean code does not have to be changed or re-compiled to run in an EJB
1.1 Container, except in the exceptions detailed below. The mandatory migration is for the
deployment descriptors to be converted to the EJB 1.1 XML.

Instances Where EJB Code Must Be Changed or Re-compiled

You must change or re-compile EJB code in the following instances:

- The bean uses the `javax.jts.UserTransaction` interface. The package name
  of the `javax.jts` interface has changed to `javax.transaction`, and there have
  been minor changes to the exceptions thrown by the methods of this interface. An
  enterprise bean that uses the `javax.jts.UserTransaction` interface needs to be
  modified to use the new name `javax.transaction.UserTransaction`.

- The bean uses the `getCallerIdentity()` or `isCallerInRole(Identity
  identity)` methods of the `javax.ejb.EJBContext` interface. These methods
  were deprecated in EJB 1.1 because the class `java.security.Identity` is
  deprecated in Java 2 platform. An enterprise bean written to the EJB 1.0 specification
  needs to be modified to use the new methods to work in *all* EJB 1.1 Containers.

- The bean is an entity bean that uses the `UserTransaction` interface. In EJB 1.1, an
  entity bean must not use the `UserTransaction` interface.

- The bean uses the `UserTransaction` interface and implements the
  `SessionSynchronization` interface at the same time. This behavior is
  disallowed in EJB 1.1.

- The bean violates any of the additional semantic restrictions defined in EJB 1.1 but
  which were not defined in EJB 1.0.

- For every EJB `Create` there must be a matching `EJBPostCreate`.
  `EJBPostCreate` was optional in EJB 1.0 and is now mandatory

Exception Handling Changes

The EJB 1.1 specification of exception handling preserved the rules defined in the EJB 1.0 specification, with the following exceptions:

- EJB 1.0 specified that the enterprise bean business methods and container-invoked callbacks use the `java.rmi.RemoteException` to report non-application exceptions. This practice is deprecated in EJB 1.1—the enterprise bean methods should use the `javax.ejb.EJBException`, or other suitable `RuntimeException` to report non-application exceptions.

- In EJB 1.1, all non-application exceptions thrown by the instance result in the rollback of the transaction in which the instance executed, and in discarding the instance. In EJB 1.0, the Container would not rollback a transaction and discard the instance if the instance threw the `java.rmi.RemoteException`.

- In EJB 1.1, an application exception does not cause the Container to automatically rollback a transaction. In EJB 1.0, the Container was required to rollback a transaction when an application exception was passed through a transaction boundary started by the Container. In EJB 1.1, the Container performs the rollback only if the instance have invoked the `setRollback-Only()` method on its `EJBContext` object.`javax.ejb.ejbex`.

EJB Deployment

EJB 1.1 has introduced the use of XML files to replace the deployment descriptor used in NAS 4.0. The Property descriptor files in your NAS 4.0 application must be converted to XML files. In addition to registering the application you must run `ejbReg`.

To convert the Property files to XML use the supplied tool `convertPropsXML`. The following steps outlines its use:

1. Convert property files to XML files.

    Use the following tool to convert the .props files to XML:.

    ```
    convertProps2Xml $path/foobar.props $newpath/myAppEjb.xml
    ```

    `$path` points to the location of `.props` file and the tool results in the generation of two XMLs `myAppEJB.xml` and `ias-myAppEjb.xml` files. These files represent the J2EE and the iPlanet Application Server-specific XML respectively.

2. Convert NTV descriptor files and add them to the EJB JAR archive file.

    a. Create a new EJB JAR module, as described in Chapter 2 of the *Administration & Deployment Guide.*

      b. On the EBJ menu, select "Import from 4.0."

      c. Navigate to the `.properties` file that you want to convert and choose OK. The `.properties` file will be converted into a `.xml` files, which will be added to your EBJ JAR module.

      d. Continue adding `.class` files and other files to your EBJ JAR module.

      e. Save and deploy your EBJ JAR module as described in Chapter 2 of the *Administration & Deployment Guide.*

3. Modify you application code to handle exceptions and transactions.

If you are using Transactions or Exceptions you may need to make some code changes. See the section "Exception Handling Changes."

4. Deploy your application as described in the *Administration & Deployment Guide*.

JNDI Migration

iPlanet Application Server 6.0 uses version 1.2 of the Java Naming and Directory Interface (JNDI) extension. JNDI is provided as part of the Java Enterprise API set.

NAS 4.0 applications that use JNDI 1.1 must be migrated to JNDI 1.2. The specific incompatibilities can be seen at:

    http://java.sun.com/products/jndi/1.2/compat.html#incompat

In J2EE every application defines its own naming environment which is specified via the component's deployment descriptor. A component's descriptor should also contain information about all EJBs and data sources that it is looking up via ejb-ref and resource-ref elements. Migrating NAS4.0 applications to iPlanet Application Server 6.0 involves the following requirements:

- Identifying usage of beans/resources that are being looked up by the application.

- Setting up appropriate deployment descriptor entries for resource-refs and ejb-refs.

- Ensure that environment lookup happens via `java:comp/env/<envionmentEntryName>` pattern as specified by the J2EE specification.

Java Extensions

To migrate NAS 4.0 Java extensions to iPlanet Application Server 6.0, perform the following steps:

1. Load IDL code in iPlanet Extension Builder 6.0 and create new generated code.

2. Merge any changes which have been made to the previous extensions into the new generated code.

3. Convert any references to NMI to JNI (if applicable).

4. Perform all other Java code changes for JDK 1.2.2.

5. Recompile all code.

C++ Extensions

To migrate NAS 4.0 C++ extensions to iPlanet Application Server 6.0, recompile and link NAS 4.0 C++ extensions against iPlanet Application Server 6.0 libraries.

Using Rich Client (ISecurity Interface)

The `ISecurity` interface needs to be implemented when using Rich Client. ISecurity sets the user name and password in Rich Client.

Security Features

iPlanet Application Server 6.0 implements security constraints at deployment time. Standard declarative access control rules are defined by the developer when the application is deployed. Developers will, for example, specify several levels of security such as administrator, guest, member etc. Then they will write code to check the current user's permission level when accessing secure procedures. At deployment time, groups of users are assigned the correct security level allowing the application to easily verify permission level before accessing the restricted procedure.

In NAS 4.0, security was implemented at the application level by setting up access control lists that define permissions granted to specific users and groups. NAS 4.0 implemented security at the code level, iPlanet Application Server 6.0 implements security independent of the code at the deployment of the application.

For tips on security go to:

http://www.java.sun.com/security/seccodeguide.html.

For EJB related security go to:

http://www.java.sun.com/j2ee/j2sdkee/techdocs/guides/ejb/html/Security.fm.html

Migration Example "The Bank"

Use this example to walk through the migration process. You can also modify the NAS 4.0 nsOnline Bank source and migrate the sample to iPlanet Application Server 6.0 prior to migrating your own applications, as an example. Refer to the code source online at http://www.iPlanet.com/support/.

This section describes the guidelines for porting a bank sample application from NAS 4.0 to iPlanet Application Server 6.0.

The iPlanet Application Server 6.0 Bank Sample application includes the following features:

- Bank sample application is using new Form based login (J2EE style).

- Proprietary methods like `HttpSession2`, `loginSession`, `NASRowSet` are replaced with the J2EE equivalent.

- EJB deployment descriptors are described in `.xml` files.

- EJB lookup is URL based like "java:comp/env/"lookupname".

- Servlet deployment is described in an `.xml` file instead of proprietary `.ntv` file.

Comparison of iPlanet Application Server 6.0 Bank Application & NAS 4.0 nsOnlineBank

This section compares NAS 4.0 and iPlanet Application Server 6.0 components.

Component	iPlanet Application Server 6.0	NAS 4.0
Servlets	Uses `HttpSession` (standard java API), for Session creation; Form-based Login mechanism (J2EE style) for Login Authentication. Bank does not use proprietary methods (e.g., Nestcape-APIs) `HttpSession2`, `loginSession`. Note that `HttpSession2` is used in iPlanet Application Server 6.0 to share session data between Applogics and servlets.	Uses `HttpSession2` method `loginSession` API for (login) authentication and also shared sessions between applogics and servlets.

Component	iPlanet Application Server 6.0	NAS 4.0
Servlet Deployment	Deployment instructions are described in xml file. Each servlet has both J2EE XML and iPlanet Application Server specific XML. Servlets are registered through `webappreg` command line utility.	Deployment instructions are described in ntv file.
EJBs	J2EE specific URL style lookup `"java:comp/env/< lookup name >"`. By getting a `DataSource` object using this lookup, Database connection is created uses new factory class `"com.netscape.server.jndi.RootContextFactory"`.	Uses old style `"ejb/<lookup name >"`. `DataSourceName` is grabbed from NAS registry, passes to `NASRowSet` method for creation of Database connection, and uses Factory class `"com.kivasoft.eb.jndi.GDSInitContextFactory"`.
EJB Deployment	Deployment descriptors are described in XML. Each EJB has both J2EE XML and iPlanet Application Server-specific XML. EJB's are registered through `ejbreg` command line utility	Deployment instructions are described in property file.
JSPs	Context Root starts from /APPS/App-Name/directory.	Context Root starts from /APPS/directory.
LDAP	Authenticate domain is " uid=admin, ou=Administrators, ou=TopologyManagement, o=NetscapeRoot"; Authenticate Password is "<user selected password at the time of installation >".	Uses default Authenticate domain (uid) "cn=Directory Manager"; Authenticate Password is "dmanager".

General Porting Guidelines

This section outlines some general porting guidelines for migrating NAS 4.0 application components to iPlanet Application Server 6.0.

- **LDAP:** LDAP code (.java file) should reflect the LDAP server port number which is supplied at installation time

- **DataBase:** DataSource should be passed through the <resource ref > tag instead of <env entry> tag.

- **Servlet:** If a servlet is doing a look up for an EJB, the corresponding J2EE Servlet XML should have `<ejb-ref>` tag.

- **For Form-based mechanism:** J2EE Servlet XML should have the `<login-config>` tag.

- **Login Page:** should be a `.jsp` file for Form-based login mechanism.

- **Text Fields for Form-based Login page:** `username` should be `"j_username"` password should be `"j_password"`.

- **Login pages:** (like `login`, `jsp`, and `loginerror.jsp`) should be kept under *`<install-location>/<app-name>.`*

- **EJBs:** if `DataSource` is passed through `<resource ref>` tag, the corresponding iPlanet Application Server EJB XML should have `<jndi-name>`, similar to the datasource `"jdbc/LocalDS"`. LocalDS.

  EJB XML files are generated using `PropToEJB` tool (note that there may be additional manual steps after the conversion).

Further Reading

Java™ 2 Platform, Enterprise Edition Specification Version 1.2 Copyright 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/j2ee/docs.html`

Java™ 2 Platform, Enterprise Edition Technical Overview (J2EE Overview).Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/j2ee/white.html`

Java™ 2 Platform, Standard Edition, v1.2.2 API Specification (J2SE specification). Copyright 1993-99, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jdk/1.2/docs/api/index.html`

Enterprise JavaBeans™ Specification, Version 1.1 (EJB specification). Copyright1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/ejb`

Enterprise JavaBeans™ to CORBA Mapping, Version 1.1 (EJB-CORBA mapping).Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/ejb`

JavaServer Pages™ Specification, Version 1.1 (JSP specification). Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jsp`

Java™ Servlet Specification, Version 2.2 (Servlet specification). Copyright 1998,1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/servlet`

JDBC™ 2.0 API (JDBC specification). Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jdbc`

JDBC™ 2.0 Standard Extension API (JDBC extension specification). Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jdbc`

Java™ Naming and Directory Interface 1.2 Specification (JNDI specification). Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jndi`

Java™ Message Service, Version 1.0.2 (JMS specification). Copyright 1998, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jms`.

Java™ Transaction API, Version 1.0.1 (JTA specification). Copyright 1998, 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jta`

Java™ Transaction Service, Version 0.95 (JTS specification). Copyright 1997-1999, Sun Microsystems, Inc. Available at `http://java.sun.com/products/jts`

JavaMail™ API Specification Version 1.1 (JavaMail specification). Copyright 1998, Sun Microsystems, Inc. Available at `http://java.sun.com/products/javamail`

JavaBeans™ Activation Framework Specification Version 1.0.1 (JAF specification). Copyright 1998, Sun Microsystems, Inc. Available at `http://java.sun.com/beans/glasgow/jaf.html`

The Java™ 2 Platform, Enterprise Edition Application Programming Model, Copyright 1999, Sun Microsystems, Inc. Available at `http://java.sun.com/j2ee/apm`.

Running NetDynamics Applications

This chapter is a planning guide for the migration of applications built with NetDynamics.

This chapter is divided into the following sections:

*   Overview

*   Migration Planning Considerations

*   iPlanet Migration Toolbox and J2EE Assisted Take-Off (JATO)

Overview

In the past, we have suggested a number of options for migrating NetDynamics 3.x, 4.x, and 5.x applications to J2EE for iAS 6.0. Some of those options specified staged upgrades to NetDynamics 5.x and then to J2EE.

With the increasing acceptance and maturity of J2EE, and because newly available automated tools can ease the transition for applications written with older versions of NetDynamics (Versions 3.x and 4.x), we favor the migration of NetDynamics 3.x and 4.x applications directly to J2EE, rather than through an intermediate NetDynamics 5.x migration step.

This chapter discusses some of the considerations involved in planning a NetDynamics to J2EE migration effort. In particular, we look at some of the issues that may affect the work effort required to migrate applications and some factors that might even dictate that a full migration to J2EE not be performed.

In addition, this chapter introduces the **iPlanet Migration Toolbox (iMT)**, which provides automated enablement for the migration of NetDynamics applications to the iPlanet Application Server J2EE environment.

The migrated applications utilize the **iPlanet J2EE Assisted Take-Off (JATO)** application framework, which not only provides for an intuitive transition of applications from the NetDynamics application framework to J2EE, but also serves as a standalone application framework upon which future J2EE development can be based. JATO is provided to iMT customers with full ownership of the source code.

A comprehensive discussion of the iMT and JATO is beyond the scope of this chapter. Please contact your iPlanet representative when you require information regarding the iMT beyond that which is provided here.

Migration Planning Considerations

Not surprisingly, the task of determining the scope of a migration effort is quite challenging. The NetDynamics environment provided for a great deal of flexibility in development. The organization of NetDynamics applications, the development standards, individual developer styles, and custom extensions and enhancements, among other things, can significantly affect the level of effort required to migrate a particular set of NetDynamics applications to J2EE.

The iPlanet Migration Toolbox includes some basic tools to jumpstart the migration estimation effort. The **NDProjectPeeker** tool generates an inventory of NetDynamics project objects to provide an idea of the size and composition of the projects. Another tool, which is currently under development, will utilize the output of the NDProjectPeeker tool to generate basic estimates for the migration effort. The initial estimates generated using the information compiled by these tools will be improved by incorporating the analysis described in this chapter.

This section includes the following topics:

*   Migration Path

*   Migration Planning and Estimating

Migration Path

We generally recommend that older NetDynamics applications be migrated directly to J2EE for iAS 6.0, but there may be some good reasons to upgrade NetDynamics 3.x and 4.x applications to NetDynamics 5.x. Some of these reasons may include:

*   It can be determined that the upgrade to NetDynamics 5.x can be performed relatively quickly for a particular application and the application will be retired prior to the termination of support for the NetDynamics 5.x product in December of 2001.

Additionally, there may be some issues that would make a near-term migration to J2EE infeasible:

- An application uses custom or third party components which can only operate within a NetDynamics environment.

- An application is dependent upon a capability or feature that only NetDynamics can provide.

However, in general, we believe that a direct migration to J2EE is more desirable:

- Applications with an expected service life beyond the next two years should generally be moved to J2EE.

- The upgrade to NetDynamics 5.x can require a significant amount of effort. In some cases, the effort required to upgrade some older NetDynamics applications to NetDynamics 5.x may be of the same magnitude as an iMT-enabled migration to J2EE for iAS, especially when tasks such as testing are taken into account.

- Any dependency on NetDynamics will have to be addressed anyway since NetDynamics support will be phased out.

- The pool of NetDynamics-trained personnel will shrink over the next few years.

Migration Planning and Estimating

The iPlanet Migration Toolbox planning tools can give a useful first impression of the magnitude of the NetDynamics to J2EE migration effort, but meaningful estimates can only be generated by analyzing the unique set of factors represented by the body of the applications to be migrated. Some of the factors that should be considered in estimating the effort required to migrate NetDynamics applications include:

- The skills set and application knowledge of the migration team:

  - J2EE

  - NetDynamics

  - Domain knowledge of applications being migrated

- The number of NetDynamics projects and pages which must be migrated.

- The extent to which the NetDynamics API was used:

  - Was the NetDynamics API used extensively for coding the event methods?

  - Were existing Enterprise JavaBeans and other business logic components written to be independent of the NetDynamics API?

- The structure of the applications:

    m   Have the applications been carefully tiered into display, business, and data layers?

    m   Are custom classes which must be migrated? What is the degree of dependence of these custom classes on NetDynamics?

- The organization of the applications:

    m   Do applications consist of many, smaller NetDynamics projects, or fewer, larger projects?

    m   The extent to which different applications within the organization are interdependent.

For larger collections of NetDynamics projects, you may need to consider including some additional tasks in the analysis:

- An architectural overview

- An in-depth analysis of any custom classes and how they are used

- An close examination of a representative sample of NetDynamics projects

- A pilot migration

iPlanet Migration Toolbox and J2EE Assisted Take-Off (JATO)

This section includes the following topics:

- iPlanet Migration Toolbox

- JATO Application Framework

- iMT/JATO Community

iPlanet Migration Toolbox

The iPlanet Migration Toolbox (iMT) contains a set of tools to perform the automated phases of NetDynamics to J2EE conversions:

- **NetDynamics Extraction Tool** – This tool extracts the declarative information and the Java code from NetDynamics projects into XML description files.

- **Application Translation Tool** – This tool uses the XML description files to construct J2EE-compliant versions of the NetDynamics projects in the JATO framework.

- Other tools – The iPlanet Migration Toolbox also contains convenience tools for compiling and packaging the translated projects.

The iMT was designed to migrate NetDynamics applications to J2EE as completely as possible. Its primary function is to move the application structure to a new application environment which supports the NetDynamics constructs and capabilities. Secondarily, the iMT tries to migrate all the declarative application functionality to this new application framework. Those functions supplied by NetDynamics wizards are, in large part, migrated to J2EE.

Once the automated part of the migration process is complete, it is necessary to evaluate the appropriateness of the custom code from the old applications. There is no practical way to convert custom code in a NetDynamics application to make it correct in a J2EE environment. The original code is commented out and moved to the appropriate module and method to make manual conversion of the code easier. Though the task is not small, it should be, in general, straightforward once the migration developer is familiar with the JATO J2EE component classes and methods.

Typically, a migration developer will perform a preliminary run of the iMT on an application to assess the level of manual migration effort that will be required for the project. The iMT generates an inventory of code-level items of interest which will need evaluation and possible modification during the manual code porting phase. This change inventory should be used to help refine the migration work plan.

JATO Application Framework

J2EE Assisted Take-Off (JATO) is an application framework built to provide developers with a foundation for building J2EE applications in a consistent and efficient manner. Instead of inventing ways to build applications and the infrastructure to support them, we have, with JATO, the procedures and support infrastructure to start building applications right away. In other words, we regain some of the productivity advantages we had with NetDynamics. Furthermore, the availability of a standard framework will not only enable organizations to ensure development consistency in the organization over time, but also reduce the likelihood that new team members would have to be trained to use a proprietary, internal J2EE application framework.

JATO is intended to not merely be a stepping-stone for the jump from NetDynamics to J2EE, but to be a framework upon which J2EE applications can be built long after the last NetDynamics application has been migrated.

iMT/JATO Community

The iPlanet Migration Toolbox team moderates an online iMT/JATO discussion community at: http://www.egroups.com/group/iPlanet-JATO

Index

S

T

U