Migration Guide

iPlanet Application Server

Version 6.0

866-3495-01
May 2000

Printed in the United States of America 00 99 98 5 4 3 2 1

SERAIENT CONTRAIRES À LA LOI.

Contents

Preface **7**
Using the Documentation 7
About This Guide 10
How This Guide Is Organized 10
Documentation Conventions 11
Related Information 11

Chapter 1 Migration Overview **13**
The New J2EE Programming Model 13
Presentation Logic and Layout 15
Business Logic 16
Data Access Logic 16
Migration Basics NAS 2.1 to iAS 6.0 17
Migration Basics NAS 4.0 to iAS 6.0 17

Chapter 2 Running NAS 2.1 Applications on iAS 6.0 . . . . . **19**
NAS 2.1 Application Components 19
HTML Templates 19
AppLogics 20
Database Logic: DAE and JDBC 20
The NAS Registry 20
Deploying NAS 2.1 Applications on iAS 6.0 21
C++ Applications and Extensions 21
Java Extensions 21
Beginning the Migration Process 21
Online Bank Example 22
US Population C++ Sample Application 23
US Population Java Sample Application 24
Migrating from NAS 2.1 to iAS 6.0 24

Chapter 3 Reimplementing Your NAS 2.1 Application 27
Redesigning Your Application 27
Migrating Presentation Logic 28
Recreating AppLogics as Servlets 28
Recreating Presentation Layout 30
Recreating Sessions and Security 30
Migrating Business Logic 31
Migrating Data Access Logic 31
Incompatibility Errors 31
Partial Migration 32
Calling EJBs from Java AppLogics 32
Calling Servlets from Java AppLogics 33
Calling Java AppLogics from Servlets 35
Calling C++ AppLogics from Servlets 36
Sessions in Partially Migrated Applications 37
Converting ITemplateData to ResultSet 38

Chapter 4 Running NAS 4.0 Applications on iAS 6.0 39
Overview 39
Differences between NAS 4.0 and iAS 6.0 40
Migrating components Step by Step 40
JDK Migration 41
JDK Migration Steps 41
Servlet Migration 42
Servlet API Changes 43
Servlet Migration Steps 44
Servlet Deployment 45
JSP Migration 46
Deprecated 46
JSP Migration Steps 46
Converting JSP 0.92 Pages to JSP 1.1 46
EJB Migration 49
EJB Migration Steps 49
Instances where EJB code must be changed or re-compiled 50
Exception Handling Changes 50
EJB Deployment 51
JNDI Migration 52
Rich Client 52
Java Extensions 53
C++ Extensions 53
Security Features 53
Migration Example “The Bank” 54
Further Reading 57

Chapter 5 NetDynamics to iAS 6.0 59
Overview 59
The Need to Migrate 59
Scoping Migrations 60
Migration Paths 62
Costs / Benefits of Migration Steps: 63
Product Support 66
NetDynamics 4 Generation Products 66
NetDynamics 5 Generation Products 66

Index 67

Preface

This preface describes the iPlanet Application Server documentation set and illustrates what you can expect to find in this *Migration Guide.*

This preface contains the following sections:

- Using the Documentation
- About This Guide
- 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 (iAS) and iPlanet Application Builder (iAB) printed manuals and online read-me file. 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 as online files in PDF and HTML format.

For information about	See the following	Shipped with
Late-breaking information about the software and the documentation	`readme.htm`	iAS 6.0, iAB 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*	iiAS 6.0

For information about	See the following	Shipped with
Installing iPlanet Application Builder.	`install.htm`	iAB 6.0
Basic features of iAS, such as its software components, general capabilities, and system architecture.	*Overview*	iAS 6.0, iAB 6.0
Administering one or more application servers using the iPlanet Application Server Administrator Tool to perform the following tasks:	*Administration & Deployment Guide*	iAS 6.0
Deploying applications with the Deployment Manager tool		
Monitoring and logging server activity		
Setting up users and groups		
Administering database connectivity		
Administering transactions		
Load balancing servers		
Managing distributed data synchronization		
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*	iiAS 6.0, iAB 6.0

For information about	See the following	Shipped with
Creating iAS 6.0 applications within an integrated development environment by performing the following tasks:	*User's Guide*	iAB 6.0
Creating and managing projects		
Using wizards		
Creating data-access logic		
Creating presentation logic and layout		
Creating business logic		
Compiling, testing, and debugging applications		
Deploying and downloading applications		
Working with source control		
Using third-party tools		
Creating iAS 6.0 applications that follow the new open Java standards model (Servlets, EJBs, JSPs, and JDBC), by performing the following tasks:	*Programmer's Guide (Java)*	iiAB 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		

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

About This Guide

This guide describes how to migrate applications from Netscape Application Server 2.1 and 4.0 to iAS 6.0. Net Dynamics instructions is covered here and online at http://iplanet.com/support

How This Guide Is Organized

This guide is organized into three chapters, as follows:

- Chapter 1, Migration Overview
- Chapter 2, Running NAS 2.1 Applications on iAS 6.0
- Chapter 3, Reimplementing Your NAS 2.1 Application
- Chapter 4, Running NAS 4.0 Applications on iAS 6.0
- Chapter 5, NetDynamics to iAS 6.0

In addition, there is a complete code walkthrough of an example migration using the Online Bank sample application from NAS 4.0 to 6.0. This is 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, emphasis, variables and placeholders, and words used in the literal sense.

Related Information

Specifications related to the iAS 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 iAS. See the index listed above for links to specification versions supported by iAS.

Servlets	`http://java.sun.com/products/servlet`
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, we recommend the following resources:

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

Chapter 1

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 applications to the new model.

The new iAS 6.0 programming model is for Java applications only. C++ applications continue to use the NAS 2.1 model. Note that:

- iAS 6.0 is backward compatible with NAS 2.1. applications. NAS 2.1 applications can run on iAS 6.0 without code alteration.
- iAS 6.0 is compatible with NAS 4.0 applications with conversion to the J2EE standard. NAS 4.0 applications do need some conversion.

The New J2EE Programming Model

The iAS 6.0 Application Server is Java 2 Platform, Enterprise Edition specification version 1.2 (J2EE 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	iAS 6.0 component	Description
Presentation Logic	AppLogic	Java servlet and propietrary 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 offloaded to EJBs, and output is usually offloaded to JSPs.
Presentation Layout (part of Presentation Logic)	HTML template	JavaServer Page (JSP) and propietrary 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.
Business Logic	AppLogic	Enterprise JavaBeans (EJBs) and propietrary 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 propietrary 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.

A Note on Modularity and Flexibility

The terms "normally" and "usually" appear frequently in this document and in the *Programmer's Guide* with regard to the roles of iAS 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 an AppLogic, while layout was handled by an HTML template. At run-time, the AppLogic provided output to populate the template.

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

Migration Basics NAS 2.1 to iAS 6.0

Migration involves altering an application written for the 2.1 programming model so that it conforms to the 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 on iAS 6.0

- **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.

  iAS 6.0 supports partial migration by providing “glue” between the old components and the new components. This support is described in .“Partial Migration” in Chapter 3, “Reimplementing Your NAS 2.1 Application

- **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, “Reimplementing Your NAS 2.1 Application

Migration Basics NAS 4.0 to iAS 6.0

Migrating to Standard. NAS 4.0 uses Netscape and older Java standards which have been replaced with J2EE 1.2 standards in iAS 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. See Chapter 4 -Running NAS 4.0 Applications on iAS 6.0.

Chapter 2

Running NAS 2.1 Applications on iAS 6.0

This chapter describes how to run NAS 2.1 applications on iAS 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 iAS 6.0. The Bank example will help you walk through a step by step migration.

iAS 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 iAS 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 21). Also, you must have correct the class path for the version of JDBC you use.

NAS 2.1 Application Components

This section describes the iAS 6.0 support for each of the major types of components from the 2.1 programming model.

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 iAS 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 iAS 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

- The new JDBC layer

  or

- 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 iAS 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
```

The NAS Registry

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

Deploying NAS 2.1 Applications on iAS 6.0

Use the iAS Administrator Tool (iASAT) to deploy all applications to iAS. For more information, see the *Administration & Deployment Guide*.

C++ Applications and Extensions

iAS 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 iAS 6.0 libraries.

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

Java Extensions

Migration of Java NAS 2.1 extensions to iAS 6.0 is as follows:

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

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, etc.) 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 Migration” in Chapter 3, “Reimplementing Your NAS 2.1 Application

Online Bank Example

Migration From NAS 2.1 to iAS 6.0 - Solaris Only

NOTE: iAS 6.0 will have JDK 1.2.2 as part of its distribution. With the new JDK1.2 all the java core packages like java.io and java.lang etc. are packaged into rt.jar instead of the classes.zip as in JDK 1.1. you can find rt.jar in /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 which is there in /iAS6.0-install-directory/nas/usr/java/bin directory for compilation

Step 1: Copy the OnlineBank java Application package to the machine where iAS 6.0 has been successfully installed by creating a new directory OnlineBank under GXAPP

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

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

/Netscape/Suitespot/docs/GXApp/OnlineBank

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

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

Step 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 iAS 6.0 kreg and not the old 2.1 version.

Step 5: Run the sample application http://hostname/GXApp/OnlineBank/OBLogin.html

and go further to see the account balance and other options.

C++ COnlineBank(NAS2.1) Sample Application

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

Step 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

Step 3: Set the following two environment variables.
setenv GX_ROOTDIR /iAS6.0-install-directory/nas
setenv GX_ROOT /iAS6.0-install-directory/nas

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

Step 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). It will copy the generated libCOnlineBank.so file into the /iAS6.0-install-directory/nas/gxlib directory.

Step 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 iAS 6.0 kreg and not the old NAS 2.1 version.

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

US Population C++ Sample Application

Migrating from NAS 2.1 to iAS 6.0

Step 1: Copy the US Population C++ application to the machine where IAS 6.0 has been successfully installed. Copy it to a new directory called CStates under GXAPP .e.g. /IAS6.0-install-directory/nas/APPS/GXApp/CStates

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

/Netscape/Suitespot/docs/GXApp/CStates

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

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

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

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

Step 6: Register using kreg utility

Change directory to /IAS6.0-install-directory/nas/APPS/GXApp/CStates then run /IAS6.0-install-directory/nas/bin/kreg states.gxr

* Make sure that you are running the IAS6.0 kreg and not the older 2.1 version.

Step 7: Run the US Population application

http://hostname/GXApp/CStates/index.html. Click on RunRegionReport to see the population statistics

US Population Java Sample Application

Migrating from NAS 2.1 to iAS 6.0

Step 1: Copy the US Population java application package to a new directory called States on the machine where iAS 6.0 has been successfully installed. .e.g GXAPP/IAS6.0-install-directory/nas/APPS/GXApp/States

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

/Netscape/Suitespot/docs/GXApp/States

and Copy all the HTML files from the US Population application in NAB 2.1 webserver docs directory into the above named States directory.

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

Step 4: Register using kreg utility

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 the IAS6.0 kreg and not the old 2.1 version.

Step 5: Run the US Population application
http://hostname/GXApp/States/index.html click on RunRegionReport to see the population statistics

Chapter 3

Reimplementing Your NAS 2.1 Application

This chapter describes altering your NAS 2.1 application to fit the iAS 6.0 programming model.This chapter contains the following sections:

- Redesigning Your Application
- Migrating Presentation Logic
- Migrating Business Logic
- Migrating Data Access Logic
- Partial Migration

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:

- 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 (withdrawals, transfers, etc.).
- 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 Migration” on page 31.

Migrating Presentation Logic

This section describes the following concepts:

- Implementing the presentation logic from old AppLogics as servlets (see “Recreating AppLogics as Servlets” on page 28)
- Converting HTML templates to JSPs (see “Recreating Presentation Layout” on page 30)
- Revising sessions and security (see “Recreating Sessions and Security” on page 30)

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 *Programmer’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 iAS creates an `IValList` member variable to contain incoming data for an AppLogic, for servlets iAS 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.

For example:

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 *Programmer’s Guide (Java)*.

Recreating Presentation Layout

In a sense, your 2.1 HTML templates are already migrated. The iAS 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 *Programmer’s Guide (Java)*.

Recreating Sessions and Security

iAS 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 *Programmer’s Guide (Java)*.

Migrating Business Logic

Business logic is handled in iAS 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 beans 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 *Programmer’s Guide (Java)*.

Migrating Data Access Logic

This section describes redeploying database calls using the JDBC API.

The JDBC layer in iAS 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 *Programmer’s Guide (Java).*

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

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 *iAS 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`), do something like this:

```
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, i.e. using `NewRequest()`, do something like this:

```
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, do something like this:

```
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 *iAS 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 iAS 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 33 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 iAS 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 *Programmer’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:

```
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 *Programmer’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 provides methods for iterating through rows in a set of memory-based hierarchical data and retrieving column values. This functionality is not supported in iAS 6.0, although group names are supported (and required).

In iAS 6.0, ITemplateData functionality is replaced with JDBC ResultSet objects. You can `convert ITemplateData` objects to ResultSets using the method `convertITemplateDataToResultSet()` from the BaseUtils class. For specific usage information, see the documentation for the BaseUtils class in the *iAS 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);
```

Chapter 4

Running NAS 4.0 Applications on iAS 6.0

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

This chapter contains the following:

- Differences between NAS 4.0 and iAS 6.0
- Migrating components Step by Step
- Migration Example "The Bank"

Overview

iAS 6.0 is certified compliant with Java 2 Platform, Enterprise Edition specification version 1.2 (J2EE 1.2). While the architecture of iAS 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 will depend on how much your applications depend on deprecated J2EE and NAS proprietary methods. Deployment and Java Server Pages require conversion procedures. Tools are provided for this. 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 below 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 iAS 6.0 available.

Differences between NAS 4.0 and iAS 6.0

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

Component	NAS 4.0	Migration Tactic	Effort	iAS 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 components Step by Step

The basic migration steps are:

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 iAS deployment tool as described in Chapter 2 of the *Administration & Deployment Guide*

JDK Migration

iAS 6.0 uses the Java 2 Development Toolkit version 1.2.2 (JDK 1.2.2).

An important difference for iAS is that if a native interface is used such as NMI, this needs to be replaced with JNI.

To better understand 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:

1. Get the list of deprecated methods and their replacements at `http://java.sun.com/j2ee/j2sdkee/techdocs/api/deprecated-list.html` and specific incompatibilities at `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

iAS 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 iAS 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 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
- The introduction of response buffering
- The introduction of distributable servlets
- The ability to get a `RequestDispatcher` by name
- The ability to get a `RequestDispatcher` using a relative path
- Internationalization improvements
- Many clarifications of distributed servlet engine semantics
- Behaviour of Servlet parameter validation has changed. See the programmers guide for more information.

Servlet API Changes

- The `getLocale` method was added to the `ServletRequest` interface to aid in determining what locale the client is in.
- i`sSecure` 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 iAS 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 method 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 do this.

OPTIONAL Step 1. 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

Step 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.

- Remove the following deprecated HTTPSession2 security methods:
```
boolean loginSession(String user, String paswd);
void logoutSession();
boolean isAuthorized(String target, String permission);
```
- Use the auth method tag in XML files to set the authentication method to either Basic, Certificate or Form based
- 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.
- Remove the ACL entries .gxr files.

Step 3. Replace URI naming that uses the AppPath as the root for absolute references to JSPs or other servlets.
In iAS 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(“/foo.jsp”);
rd.forward(req,res);
```

Here `foo.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.

Step 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 2 new files, myApp.xml, and ias-myApp.xml, in $newpath. These represent the J2EE and the ias-specific XML respectively.

Step 2: Convert NTV decriptor files and add them to the EJB JAR archive file.

a. Create a new web application as described in Chapter 2 of the *Administration & Deployment Guide.*

b. On the Servlet menu, select “Import from 4.0”.

c. Navigate to the appInfo.ntv file that you want to convert and choose OK. The appInfo.ntv file and servlet files that it points to will be converted into a .xml files, which willbe added to yourweb application.

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

e. Save and deploy your web application as described in Chapter 2 of the *Administration & Deployment Guide.*

Step 3: Deploy your application as described in Chapter 2, “Running NAS 2.1 Applications on iAS 6.0.

JSP Migration

iAS 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 specification includes the following primary changes from JSP 0.92 to JSP 1.1:

JSP 1.1 extends JSP 0.92 and 1.0 by:

- 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 iAS 6.0 provides caching and load balancing for JSPs and provides custom tag extensions.

Deprecated

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

JSP Migration Steps

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

Step 2. Convert Java Server Pages from specification 0.92 to 1.1
Java Server Pages must be migrated. Follow the instructions below

Converting JSP 0.92 Pages to JSP 1.1

Use the supplied conversion tool 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.

CAUTION BE SURE 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
```

-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: In these examples, `appPath` is `C:\Netscape\Server4\NAS\APPS` on Windows NT and `/export/nas4/nas/APPS` on Solaris.)

To convert 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

To convert 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

iAS 6.0 employs version 1.1 of the Enterprise Java Beans (EJB) Specification. For detailed information on the Specification go to http://java.sun.com/products/ejb/

The 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 other changes in the EJB 1.1 specification were made to improve the support for the development, application assembly, and deployment of iAS 6.0 applications.

Major Changes

- 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

Step 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.

Step 2. Convert deployment descriptors
See section on EJB Deployment

Step 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

- 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 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.`

Step 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 represent the J2EE and the iAS-Specific XML respectively.

Step 2: Convert NTV decriptor 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 willbe 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.*

Step 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”.

Step 4: Deploy your application as described in the Deployment manual.

JNDI Migration

iAS 6.0 uses version 1.2 of the Java Naming and Directory Interface 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 iAS6.0 involves the following:

- 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.

Rich Client

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

Java Extensions

Migration of Java NAS 4.0 extensions to iAS 6.0 is as follows:

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

Recompile and link NAS 4.0 C++ Extensions against iAS 6.0 libraries

Security Features

iAS 6.0 implements security constraints at deployment time. Standard declarative access control rules are defined by the developer when the application is deployed. The developer will, for example, specify several levels of security such as administrator, guest, member etc. Then she 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, iAS 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 iAS 6.0 prior to migrating your own applications, as an example. Refer to the source online at http://www.iPlanet.com/support/.

This describes the guide lines for porting a bank sample application from NAS 4.0 to iAS 6.0

Features of the iAS 6.0 Bank Sample Application

- 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 xml file instead of proprietary "ntv" file

Comparison of iAS 6.0 Bank Application & NAS 4.0 nsOnlineBank

Components	**iAS 6.0**	**NAS 4.0**
Servlets	UsesHttpSession (standard java API), for Session creation; FormBased Login mechanism (J2EE style) for Login Authentication Bank does not use proprietary methods (Nestcape-APIs like) HttpSession2,loginSession . Note that HttpSession2 is used in iAS 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.
Servlet Deployment	Deployment instructions are described in xml file. Each servlet has both J2EE XML and iAS specific XML. Servlets are registered through webappreg command line utility	Deployment instructions are described in ntv file
EJB	J2EE specific URL style lookup "java:comp/env/< lookup name >". By getting 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. uses Factory class "com.kivasoft.eb.jndi.GDSInitContext Factory"
EJB Deployment	Deployment descriptors are described in XML. Each ejb has both J2EE XML and iAS specific XML. EJB's are registered through ejbreg command line utility	Deployment instructions are described in property file

JSP	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 Guide Lines for Bank Sample Application (from NAS 4.0 to iAS 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.
- The 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 & loginerror.jsp`) should be kept under *`<install-location>/<app-name>`*

- EJB

  If DataSource is passed through `<resource ref >` tag, the corresponding iAS EJB XML should have `<jndi-name>` like `"jdbc/LocalDS"`. LocalDS is the datasource

  EJB XML files are generated using `PropToEJB` tool (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`.

Chapter 5

NetDynamics to iAS 6.0

Overview

For those who have significant investments in the Sun/NetDynamics Application Server technology, this chapter and it's online detail provides answers to many of the questions about present and future application development paths

NetDynamics Application Server is now part of the iPlanet Application Server. Customers have different migration paths depending on need.

- Updating from previous versions of NetDynamics
- Updating to the J2EE programming Model

See the complete details of NetDynamics migration on the iPlanet website; http://www.iplanet.com/support/nd/. This chapter will act as a starting point for planning guide for decisions surrounding these migrations. This Chapter also discusses some of the issues surrounding how existing NetDynamics customers can start to take advantage of the J2EE development standard in NetDynamics 5 and iAS

The Need to Migrate

Migration to NetDynamics 5 and J2EE is something that all NetDynamics installations must consider. Migration to NetDynamics 5, even the non-J2EE version, will provide all installations with adequate time to plan further migrations. End-of-life for some earlier versions has been announced, and support for some will terminate.

Beyond the migration to the NetDynamics object framework in NetDynamics 5, J2EE is the set of standards to which most NetDynamics-type applications are moving. Starting to think now about moving to NetDynamics 5 and eventually to the J2EE standard will pay off in flexibility later.

Some of the reasons to move immediately to NetDynamics 5 or iPlanet:

- bug fixes
- product enhancements
- performance improvements - NetDynamics 5.x offers good performance improvements, with some sites reporting over 100% speedup.
- simplified administration - The web server plugin system has been rewritten, is simpler to configure, and now only uses one fixed port to communicate with the application server for simplified firewall configuration.
- server AND studio support for EJBs and their generation
- support for J2EE for full or partial migrations - including Servlets and JSPs
- continued availability of technical support until the end of 2002
- complete product install available in the first versions of the iPlanet application server

Scoping Migrations

Tasks that should be considered to determine whether migration is necessary or desirable and the scope of the migration effort include:

- An architectural overview, including the use of object tiers and direct object binding
- An overview of any custom classes or NetDynamics framework classes
- An examination of a representative sample of the NetDynamics projects
- A pilot migration

These are some examples of some issues that should be addressed (along with the timelines and resources required) when scoping migrations to NetDynamics 5 and beyond to J2EE:

- The expected life span of the applications
- The number of NetDynamics projects and pages which must be migrated
- The extent to which the NetDynamics class libraries are used
- Have the developed applications been carefully tiered into display, business, and data layers?
- What is the number of custom and base classes which must be migrated (and the breakdown of these which are NetDynamics-specific and those that are not)?
- Does the current development process rely heavily on a framework developed to augment or mask the base product's API?
- The extent to which different applications within the organization are interdependent
- How the use of proprietary features of NetDynamics could be addressed in a more standardized fashion
- Whether existing EJBs and other business logic have been written to be independent of NetDynamics Object Framework classes
- What proprietary NetDynamics features are being used that could endanger the project if these features disappear?
- Whether features are needed which are only available via migration
- Whether features or integrated third-party components could not be used after migration
- The need for the additional features of recent JDK releases
- The need for Y2K compliance
- The need for NetDynamics Technical Support
- The need for additional performance

Migration Paths

Several different directions can be taken in moving forward with NetDynamics. The migration can be attempted as a single jump to J2EE, or there are several intermediate steps that can be used to provide waypoints and additional options for migration resources:

- NetDynamics 4
- NetDynamics 5 (non-J2EE)
- NetDynamics 5 (J2EE)
- iPlanet J2EE

Several different tactics may be employed as to the use of resources in accomplishing the migration of old and new applications:

- Migrate existing projects to NetDynamics 4 or 5 (non-J2EE) depending upon project life span estimates; continue to develop all new projects in NetDynamics 5 (non-J2EE).
- Migrate existing projects to NetDynamics 4 or 5 depending upon project lifespan estimates; new implementations in NetDynamics 5 (non-J2EE) for short-lifespan projects, other new implementations in NetDynamics J2EE.
- Migrate existing projects to NetDynamics 5 (J2EE) as a phased migration - migrate visual elements to J2EE while retaining NetDynamics methods for business logic and data access.
- New applications in, and migration of critical and long-life subprojects to iPlanet J2EE, others left in NetDynamics 5 (session and EJB cross-communication between the two will be supported)
- Migration of all projects immediately to J2EE.

Costs/Benefits of Migration Steps:

Tactic	Benefit	Cost / Limitation
ND 3 to ND 4	Supported until end of 2000 Better Development Studio CORBA Based Application Server Native support for more databases (DB2, …) Support for client-side Java Performance enhancements Bug fixes Improved JDK 1.1.x	Migration to new event model is needed for effective continued development Greater CPU and memory required Support for the NetDynamics 4 will be dropped sooner than 5 Port to JDK 1.1.x
ND 3 to ND 5	Supported until 2002- allows time for thorough investigation of J2EE New Web Server Plugin Architecture Better Development Studio, Administration Tool CORBA Based Application Server JDBC support for more databases (DB2, …) Support for client-side Java Performance enhancements Bug fixes Improved JDK 1.2	Migration to new event model is needed for effective continued development Greater CPU and memory required DS service based on JDBC, requires JDBC drivers Port to JDK 1.2

ND 4 to ND 5 (non-J2EE)		Supported until 2002 - allows time for thorough investigation of J2EE JDK 1.2 New Web Server Plugin Architecture Better Administration Tool Trivial upgrade for many installations Performance Enhancements Bug fixes	DS service based on JDBC, requires JDBC drivers Support for the proprietary API will eventually end A few version 4 PACs no longer supported Greater CPU and memory required
ND 3, ND 4, ND 5, to J2EE	General	The J2EE API is an open platform with a variety of server and development tool implementations	Pure J2EE requires migration of all code dependant on proprietary APIs Display mechanisms of JSPs/Servlets completely different paradigm to ND events Loss of rich event model and project security model Developer retraining in J2EE necessary Greater CPU and memory required
	ND 5 J2EE	Partial/phased migrations possible using J2EE and ND API Preserves training investment Preserves current deployment timelines	Support for the proprietary API will eventually end

	NAS 4 J2EE	Entity EJBs Server has no single point of failure	Migration of all code dependant on the NetDynamics API JDK 1.1.x (may need to BACK-port from NetDynamics 5 under JDK 1.2) Servlet 0.92 (quite different from 1.0) Support for the proprietary API will eventually end No support for any NetDynamics PAC's Loss of productivity of IDE and automatic data binding No interoperability between running ND and NAS apps Requires immediate re-implementation of base, utility, and framework classes
	iAS J2EE	Entity EJBs Server has no single point of failure Interoperability between running ND apps and iAS apps	Migration of all code dependant on the NetDynamics API Loss of productivity of IDE and automatic data binding Requires re-implementation of base, utility, and framework classes

The correct tactic is a decision that must take in to account not only technical matters, but product timing, existing investment in technology and training, the timeframe to get the development and deployment forces retrained, and project life spans.

Product Support

Product Support timelines for the different versions of NetDynamics must be taken into consideration in the application migration decision process. The following is a summary of the support NetDynamics offers across the various releases of its products (check the website for up-to-date availability of all platforms).

NetDynamics Product Line	End Of Technical Support	Ship date
ND 3	12/31/1999 *	Available now
ND 4	10/01/2000	Available now
ND 5	12/31/2002	Available now
ND 5 J2EE (development release)	12/31/2002	Available now
ND 5 J2EE (deployment release)	12/31/2002	Jan 2000
iAS	-	April 2000

*HP/UX and AIX versions may be available slightly longer, please check for latest information.

NetDynamics 4 Generation Products

The NetDynamics 4.0 Application Server should be migrated to NetDynamics 4.1.3 for NT and Solaris, NetDynamics 4.1.2 for HP-UX and AIX for Y2K compliance. The NetDynamics 4.1.2 Application Server is Y2K compliant with the patch and the NetDynamics 4.1.3 Application Server is Y2K compliant. All the PACs are Y2K compliant.

NetDynamics 5 Generation Products

All NetDynamics 5 generation products are Y2K compliant.

Index

A

all, 32
AppLogics, 20

B

Business Logic, 16

C

C++ Extensions, 53
Comparison of iAS 6.0 Bank Application & NAS 4.0 nsOnlineBank, 55
Complete migration to the new programming model., 17
Convert NTV decriptor files, 45
Converting, 46
Converting ITemplateData to ResultSet, 38
Converting JSP 0.92 Pages to JSP 1.1, 46
Costs / Benefits of Migration Steps , 63

D

Database Logic
DAE and JDBC, 20
Deploying NAS 2.1 Applications on iAS 6.0, 21
Deprecated, 46
documentation, 7

E

EJB JAR, 51
EJB Migration, 49
EJB Migration Steps, 49
Exception Handling Changes, 50

F

format
URLs, in manual, 11
Further Reading, 57

G

General Porting Guide Lines for Bank Sample Application (from NAS 4.0 to iAS 6.0), 56

J

J2EE Programming Model, 13
Java Extensions, 53
JDK Migration, 41
JDK Migration Steps, 41
JNDI Migration, 52
JSP Migration, 46
JSP Migration Steps, 46

M

Migration Basics NAS 4.0 to iAS 6.0, 17
Migration Paths, 62

N

NAS 2.1 Application Components, 19
NetDynamics 5 or iPlanet, 60
Netscape Application Server
  documentation, 7

P

Presentation Logic and Layout, 15
Product, 66
Product Support, 66

R

Reimplementing Your NAS 2.1 Application, 17
Rich Client, 52
Running NAS 2.1 Applications on iAS 6.0, 19

S

Scoping Migrations, 60
Security Features, 53
Servlet API Changes, 43
Servlet Deployment, 45
Servlet Migration, 42
Sessions in Partially Migrated Applications, 37

T

The NAS Registry, 20

U

URLs
  format, in manual, 11
US Population C++ Sample Application, 23
US Population Java Sample Application, 24