

# Migration Guide

Netscape Application Server

Version 4.0

Copyright © 1999 by Netscape Communications Corp., a subsidiary of America Online, Inc. All rights reserved. Netscape Communications Corporation and its licensors retain all ownership rights to the software programs developed by Netscape Communications Corporation and related documentation. Portions of the Software copyright © 1994, 1995 Sun Microsystems, Inc. All rights reserved. Use of this manual is intended for the original purchaser or assignee only. No portion of this manual maybe reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use, without the express written permission of Netscape Communications Corporation.

Netscape and Netscape Application Server are trademarks of Netscape Communications Corporation.

Netscape, Netscape Navigator, Netscape ONE and the Netscape N and Ship's Wheel logos are registered trademarks of Netscape Communications Corporation in the United States and other countries. Netscape FastTrack Server and Netscape Enterprise Server are trademarks of Netscape Communications Corporation, which may be registered in other countries. Microsoft is a registered trademark and Windows NT is a trademark of Microsoft Corporation. Sun Microsystems is a registered trademark and Solaris is trademark of Sun Microsystems, Inc. Hewlett Packard is a registered trademark and HP-UX is a trademark of Hewlett-Packard Company. Trademarks may be registered in various jurisdictions. IBM, CICS, and MQSeries are registered trademarks and IMS is a trademark of IBM Corporation. BEA and TUXEDO are registered trademarks of BEA Systems, Inc. TransFuse is a registered trademark of Insession, Inc. All other trademarks and registered trademarks contained herein are the property of their respective owners.

Notice:

The information contained in this document is subject to change without notice. THIS DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. Netscape Communications Corporation shall not be liable for error contained herein or damages in connection with performance or use of this documentation. Netscape Communications Corporation makes no warranty or representation, either express or implied, with respect to the documentation, including its quality, performance, merchantability, or fitness for a particular purpose. In no event will Netscape Communications Corporation be liable for direct, indirect, special, incidental, cover, or consequential damages, including losses, costs, charges, claim for lost profits, or fees of any nature or kind arising out of the use of or inability to use the documentation, even if advised of the possibility of such damages. Use of the software referenced in this document is governed by the terms of an executed license agreement between you and Netscape Communications Corporation. The downloading, export or reexport of the software or any underlying information or technology must be in full compliance with all United States and other applicable laws and regulations. Any provision of the accompanying software or this document to the U.S. Government is with restricted rights as described in the license agreement accompanying the software.



Recycled and Recyclable Paper

Version 2.1

Part Number 151-07591-00

©1999 Netscape Communications Corp. All Rights Reserved

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

Netscape Communications Corporation, 501 East Middlefield Road, Mountain View, CA 94043

# Contents

Using the Documentation .....	5
About This Guide .....	9
How This Guide Is Organized .....	9
Documentation Conventions .....	10
Related Information .....	11
<b>Chapter 1 Migration Overview .....</b>	<b>13</b>
The New Programming Model for Java .....	13
Presentation Logic and Layout .....	15
Business Logic .....	16
Data Access Logic .....	16
Migration Basics .....	17
<b>Chapter 2 Running NAS 2.1 Applications on NAS 4.0 .....</b>	<b>19</b>
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 NAS 4.0 .....	21
C++ Applications and Extensions .....	21
Beginning the Migration Process .....	21
<b>Chapter 3 Reimplementing Your Application .....</b>	<b>23</b>
Redesigning Your Application .....	23
Migrating Presentation Logic .....	24
Recreating AppLogics as Servlets .....	24
Recreating Presentation Layout .....	26
Recreating Sessions and Security .....	26

Migrating Business Logic .....	27
Migrating Data Access Logic .....	27
Partial Migration .....	28
Calling EJBs from Java AppLogics .....	28
Calling Servlets from Java AppLogics .....	30
Calling Java AppLogics from Servlets .....	31
Calling C++ AppLogics from Servlets .....	33
Sessions in Partially Migrated Applications .....	33
Converting ITemplateData to ResultSet .....	34

# Preface

This preface describes the NAS 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 Netscape Application Server (NAS) and Netscape Application Builder (NAB) 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.

## Using the Documentation

For information about	See the following	Shipped with
Late-breaking information about the software and the documentation	<code>readme.htm</code>	NAS 4.0 Developer Edition (Solaris), NAS 4.0, NAB 4.0
Installing Netscape Application Server and its various components (Web Connector plug-in, Netscape Application Server Administrator), and configuring the sample applications	<i>Installation Guide</i>	NAS 4.0 Developer Edition (Solaris), NAS 4.0
Installing Netscape Application Builder.	<code>install.htm</code>	NAB 4.0
Basic features of NAS, such as its software components, general capabilities, and system architecture.	<i>Overview</i>	NAS 4.0 Developer Edition (Solaris), NAS 4.0, NAB 4.0
Deploying Netscape Application Server at your site, by performing the following tasks: <ul style="list-style-type: none"><li>• Planning your Netscape Application Server environment</li><li>• Integrating the product within your existing enterprise and network topology</li><li>• Developing server capacity and performance goals</li><li>• Running stress tests to measure server performance</li><li>• Fine-tuning the server to improve performance</li></ul>	<i>Deployment Guide</i>	NAS 4.0

For information about	See the following	Shipped with
<p>Administering one or more application servers using the Netscape Application Server Administrator tool to perform the following tasks:</p> <ul style="list-style-type: none"> <li>• Deploying applications with the Deployment Manager tool</li> <li>• Monitoring and logging server activity</li> <li>• Setting up users and groups</li> <li>• Administering database connectivity</li> <li>• Administering transactions</li> <li>• Load balancing servers</li> <li>• Managing distributed data synchronization</li> </ul>	<i>Administration Guide</i>	NAS 4.0
Migrating your applications to the new Netscape Application Server 4.0 programming model from version 2.1, including a sample migration of an Online Bank application provided with Netscape Application Server	<i>Migration Guide</i>	NAS 4.0 Developer Edition (Solaris), NAS 4.0, NAB 4.0

## Using the Documentation

For information about	See the following	Shipped with
<p>Creating NAS 4.0 applications within an integrated development environment by performing the following tasks:</p> <ul style="list-style-type: none"><li>• Creating and managing projects</li><li>• Using wizards</li><li>• Creating data-access logic</li><li>• Creating presentation logic and layout</li><li>• Creating business logic</li><li>• Compiling, testing, and debugging applications</li><li>• Deploying and downloading applications</li><li>• Working with source control</li><li>• Using third-party tools</li></ul>	<i>User's Guide</i>	NAB 4.0
<p>Creating NAS 4.0 applications that follow the new open Java standards model (Servlets, EJBs, JSPs, and JDBC), by performing the following tasks:</p> <ul style="list-style-type: none"><li>• Creating the presentation and execution layers of an application</li><li>• Placing discrete pieces of business logic and entities into Enterprise Java Bean (EJB) components</li><li>• Using JDBC to communicate with databases</li><li>• Using iterative testing, debugging, and application fine-tuning procedures to generate applications that execute correctly and quickly</li></ul>	<i>Programmer's Guide (Java)</i>	NAS 4.0 Developer Edition (Solaris), NAB 4.0



For information about	See the following	Shipped with
Using the public classes and interfaces, and their methods in the Netscape Application Server class library to write Java applications	<i>Server Foundation Class Reference (Java)</i>	NAS 4.0 Developer Edition (Solaris), NAB 4.0
Creating NAS C++ applications using the NAS class library by performing the following tasks: <ul style="list-style-type: none"> <li>• Designing applications</li> <li>• Writing AppLogics</li> <li>• Creating HTML templates</li> <li>• Creating queries</li> <li>• Running and debugging applications</li> </ul>	<i>Programmer's Guide (C++)</i>	Order separately
Using the public classes and interfaces, and their methods in the Netscape Application Server class library to write C++ applications	<i>Server Foundation Class Reference (C++)</i>	Order separately

## About This Guide

This guide describes how to migrate applications from Netscape Application Server 2.1 to NAS 4.0.

## 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 NAS 4.0”
- Chapter 3, “Reimplementing Your Application”

In addition, there is a complete code walkthrough of an example migration using the Online Bank sample application from NAS 2.1. This is available online from the following URL:

`http://home.netscape.com/eng/server/nas/index.htm`

## 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 NAS programming model are provided in the directory *installdir/nas/docs/javaspec*, where *installdir* refers to the directory in which you installed NAS. You can find a directory of all NAS-related documentation at *installdir/nas/docs/index.htm*.

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 NAS. See the index listed above for links to specification versions supported by NAS.

Servlets	<a href="http://java.sun.com/products/servlet">http://java.sun.com/products/servlet</a>
JavaServer Pages (JSPs)	<a href="http://java.sun.com/products/jsp">http://java.sun.com/products/jsp</a>
Enterprise JavaBeans (EJBs)	<a href="http://java.sun.com/products/ejb">http://java.sun.com/products/ejb</a>
Java Naming and Directory Interface (JNDI)	<a href="http://java.sun.com/products/jndi">http://java.sun.com/products/jndi</a>
Java Database Connectivity (JDBC)	<a href="http://java.sun.com/products/jdbc">http://java.sun.com/products/jdbc</a>

Additionally, we recommend the following resources:

### Programming with Servlets and JSPs

*Java Servlet Programming*, by Jason Hunter, 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>

## Related Information

### Programming with JDBC

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

*JDBC Database Access With Java: A Tutorial and Annotated Reference (Java Series)*, by Graham Hamilton, Rick Cattell, Maydene Fisher

# Migration Overview

This chapter introduces the Netscape Application Server 4.0 programming model and compares it to the NAS 2.1 programming model. It also describes the basics of migrating 2.1 applications to the new model.

**Note** The new NAS 4.0 programming model is for Java applications only. C++ applications continue to use the NAS 2.1 model.

## The New Programming Model for Java

The NAS 4.0 Java programming model 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.

Application flow is similar between the 4.0 model and the 2.1 model. 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 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	Description
Presentation Logic	AppLogic	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 4.0, 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)	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)	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	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 4.0 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 NAS 4.0 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 NAS 4.0 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 NAS 4.0 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 NAS 4.0 programming model, data access logic is handled by the JDBC standard set of APIs. The previous APIs are all deprecated in NAS 4.0.

## Migration Basics

Migration involves altering an application written for the 2.1 programming model so that it conforms to the 4.0 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 NAS 4.0.”

- **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.

NAS 4.0 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 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.

## Migration Basics

This approach is described in Chapter 3, “Reimplementing Your Application.”

# Running NAS 2.1 Applications on NAS 4.0

This chapter describes how to run NAS 2.1 applications on NAS 4.0 without making any source-level changes.

NAS 4.0 is completely backward-compatible with NAS 2.1. In other words, you should be able to deploy your older NAS 2.1 application on NAS 4.0 without alteration, although C++ applications and extensions must be recompiled before deploying on the new server (see “C++ Applications and Extensions” on page 21).

## NAS 2.1 Application Components

This section describes the NAS 4.0 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 NAS 4.0, 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 *Netscape 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 NAS 4.0, 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 NAS 4.0 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: APPLAGIC-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 Guide* for more information.

## Deploying NAS 2.1 Applications on NAS 4.0

Use the NAS Deployment Manager to deploy all applications to NAS. For more information, see the *Administration Guide*.

## C++ Applications and Extensions

NAS 4.0 provides new versions of required C++ header files. For this reason, C++ applications and extensions must be recompiled using the new header files.

NAS provides pre-built extensions for several legacy systems, including MQSeries, TUXEDO, and CICS. These extensions will be re-released to provide support for NAS 4.0, 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, etc.) rather than by application component.

For information on partial migration, including how to allow 2.1 components (like AppLogics) to interact with 4.0 components (like servlets and EJBs), see “Partial Migration” in Chapter 3, “Reimplementing Your Application.”

## Beginning the Migration Process

# Reimplementing Your Application

This chapter describes altering your NAS 2.1 application to fit the NAS 4.0 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 28.

## 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 24)
- Converting HTML templates to JSPs (see “Recreating Presentation Layout” on page 26)
- Revising sessions and security (see “Recreating Sessions and Security” on page 26)

## 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 NAS creates an `IVallList` member variable to contain incoming data for an `AppLogic`, for servlets NAS instead creates a request object and passes it as a parameter to the servlet. Likewise, where `AppLogics` use an `IVallList` 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** 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 NAS 4.0 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 eventually 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

NAS 4.0 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 is based on the old-style model, wherein a component (servlet or AppLogic) logs into a session by authenticating the user. The session is then trusted, and an identity object is created (of type `java.security.Identity`). This also fits with the EJB security model,

which relies on an instance of `java.security.Identity`. 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 NAS 4.0 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, if you have an electronic shopping cart

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)*.

## Migrating Data Access Logic

This section describes redeploying database calls using the JDBC API.

The JDBC layer in NAS 4.0 supports 100% of the JDBC 1.22 specification plus selected areas of the JDBC 2.0 specification. The JDBC 2.0 areas that have been implemented are:

- ResultSet Scrolling, types `FORWARD_ONLY` and `INSENSITIVE`.
- Distributed Transaction Support
- DataSources
- RowSets

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)*.

## Partial Migration

**Note** The JDBC 2.0 interfaces provided in `$GX_ROOTDIR/solarisdbg/JDK_1.1/java` (or similar directory) must be before any other JDBC interfaces in the `CLASSPATH`. NAS 4.0 works with JDK 1.1.6, which has JDBC 1.2 interfaces in `$JAVA_HOME/lib/rt.jar`, so make sure this `rt.jar` is after the NAS 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: APPLLOGIC-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 cart 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 *NAS Foundation Class 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 servlet 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_fortune",
                                         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 *NAS 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.*;7
```

```
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 NAS 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.*;

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 Java AppLogics from Servlets” on page 31 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 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.

You also use `loginSession( )` to authenticate a session. 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 NAS 4.0, although group names are supported (and required).

In NAS 4.0, `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 *NAS 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);
```

## Partial Migration