Developer’s Guide

iPlanet Unified Integration Framework

Version 6.0

806-5506-02
October 2000

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

Contents

List of Figures

Preface

This preface contains the following sections:

- About This Guide
- Installation Sequence
- What You Should Know
- How This Guide Is Organized
- User Roles and Tasks
- Documentation Conventions
- Online Guides
- Related Information

About This Guide

This guide describes the iPlanet Application Server Unified Integration Framework (UIF) concepts, and how to configure and use the iPlanet Application Server with an iPlanet Enterprise Connector. Also described are how to write Servlets, JavaServer Pages (JSPs), and Enterprise JavaBeans (EJBs) which use the UIF Application Programming Interface (API) to provide real time access to Enterprise Information System (EIS) applications. In addition, all APIs for the UIF classes are fully documented.

Installation Sequence

This guide assumes the following applications have been installed, and are up and running:

- iPlanet Web Server
- iPlanet Application Server

What You Should Know

This guide assumes you are familiar with the following topics:

- iPlanet Application Server programming concepts
- Internet and World Wide Web
- Common Enterprise Information Systems programming concepts
- Java programming language and Java Servlets
- JavaServer Pages
- Enterprise JavaBeans

How This Guide Is Organized

This guide is organized into the following chapters:

Chapter 1, “Concepts” describes concepts to be familiar with before setting up UIF or use the API in Servlets, JSPs, or EJBs.

Chapter 2, “Installation” describes how-to install UIF 6.0 SP1 on your iPlanet Application Server.

Chapter 3, “Repository” describes the repository, and how to use the repository browser to view and modify EIS information.

Chapter 4, “Pooling Concepts” describes pooling concepts to be familiar with when setting up the repository.

Chapter 5, “Programming Model” describes the UIF Application Programing Model to be familiar with before using the UIF API in Servlets, JSPs, or EJBs.

Chapter 6, “Exceptions” describes the generic exception framework implemented in UIF including the various types, scenarios and how to handle them.

Chapter 7, “API Reference” describes the classes and methods used from Servlets, JSPs, or EJBs to access and manipulate data stored on an EIS.

“Glossary” describes the terms used in this guide.

User Roles and Tasks

Different skill sets are involved in setting up UIF, they are as follows:

System Administrator	• Installs UIF
Data Source Administrator	• Mines the data to define the EIS transactions available through UIF • Uses the management console to specify the enterprise connector's configuration, which end users have access to specific data, and performance characteristics such as pooling
Applications Programmer	• Writes J2EE components which call the UIF APIs • Uses the repository browser to determine the available data types and access methods

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 slashes are used instead of backslashes to separate directories.

This guide uses URLs of the form:

http:/ /*server.domain* / *path* / *file*.htm

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. Italics items in URLs are place holders.

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 place holders, and words used in the literal sense.

Online Guides

You can find the *iPlanet Unified Integration Framework Developer's Guide* online in PDF and HTML formats. To locate these files, use the following URL:

http://docs.iplanet.com/docs/manuals/ias.html

Related Information

In addition to this guide, the various enterprise connector comes with an administrator's and developer's guides. The administrator's guide explains how to set up users, data mining and other resources used by the enterprise connector to access EIS services which contain the business logic. The developer's guide explains how to write Java programs (Servlet / JSP / EJB) to access EIS services which contain the business logic.

The installer copies the *iPlanet Unified Integration Framework Developer's Guide* to *ias/APPS/docs/bsp* subdirectory of the root installation of the iPlanet Application Server.

In addition to these guides, there is additional information for administrators, users and developers. Use the following URL to view the related documentation:

http://docs.iplanet.com/docs/manuals/ias.html

This site includes the following documents:

- *iPlanet Unified Integration Framework Release Notes*
- *iPlanet Web Server Developer's Guide*
- The various *iPlanet Application Server Enterprise Connector Administrator's Guides (R/3, Tuxedo, PeopleSoft and CICS)*
- The various *iPlanet Application Server Enterprise Connector Developer's Guides (R/3, Tuxedo, PeopleSoft and CICS)*
- *iPlanet Application Server Installation Guide*
- *iPlanet Application Server Overview Guide*
- *iPlanet Application Server Release Notes*
- *iPlanet Application Server Administrator's Guide*
- *iPlanet Application Builder User's Guide*
- *iPlanet Application Builder Installation Guide*

- *iPlanet Application Builder Release Notes*
- *iPlanet Administration and Deployment Tool*

Chapter 1

Concepts

This chapter describes the iPlanet Unified Integration Framework (UIF) concepts. Also included are topics you need to be familiar with before setting up UIF or before using the UIF Application Programming Interface (API) in Java 2 Enterprise Edition (J2EE) components.

This chapter contains the following sections:

- About UIF
- UIF Architecture
- UIF Abstractions Overview
- UIF Tools
- UIF API
- UIF Application Development Model

Refer to the “Glossary” for descriptions of terms used in this guide.

About UIF

UIF is used to build interactive web-based applications on top of disparate enterprise data sources. Specifically, UIF allows the accessing of data on a supported Enterprise Information System (EIS) using the UIF APIs via a Servlet, a JavaServer Page (JSP) or an Enterprise JavaBean (EJB) running in an iPlanet Application Server.

The UIF APIs include functionality found traditionally on various low level EIS APIs. Consequently, application developers are able to use the UIF APIs and service descriptions in the UIF repository to bring the EIS data and services to the web sooner, since they can bypass the need to learn the low-level EIS APIs.

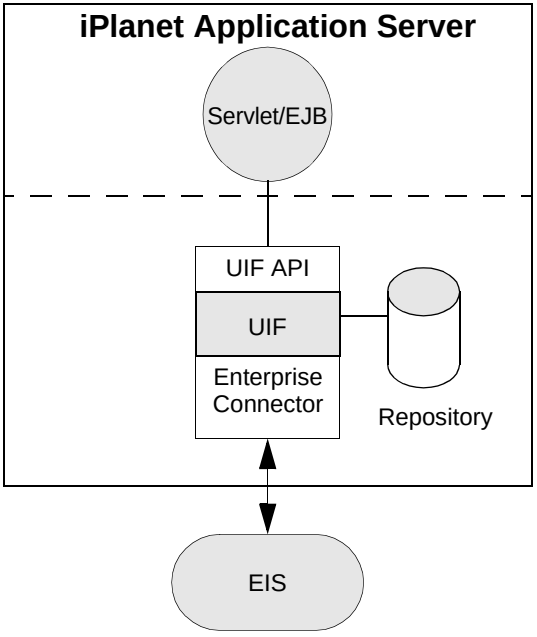
In addition, UIF supports I18N features of iPlanet Application Server. When the iPlanet Application Server is running in I18N mode, UIF allows locale and character set specific J2EE components to interact with the respectively configured enterprise data sources.

UIF Architecture

UIF is designed to be easy to use, scalable, and reliable service inside the iPlanet Application Server. This section explains the UIF architecture and UIF's role in process-based or J2EE applications.

Figure 1-1 shows the relationship between a Servlet, JSP, or EJB running in the iPlanet Application Server, UIF, and the EIS.

Figure 1-1 UIF Overview

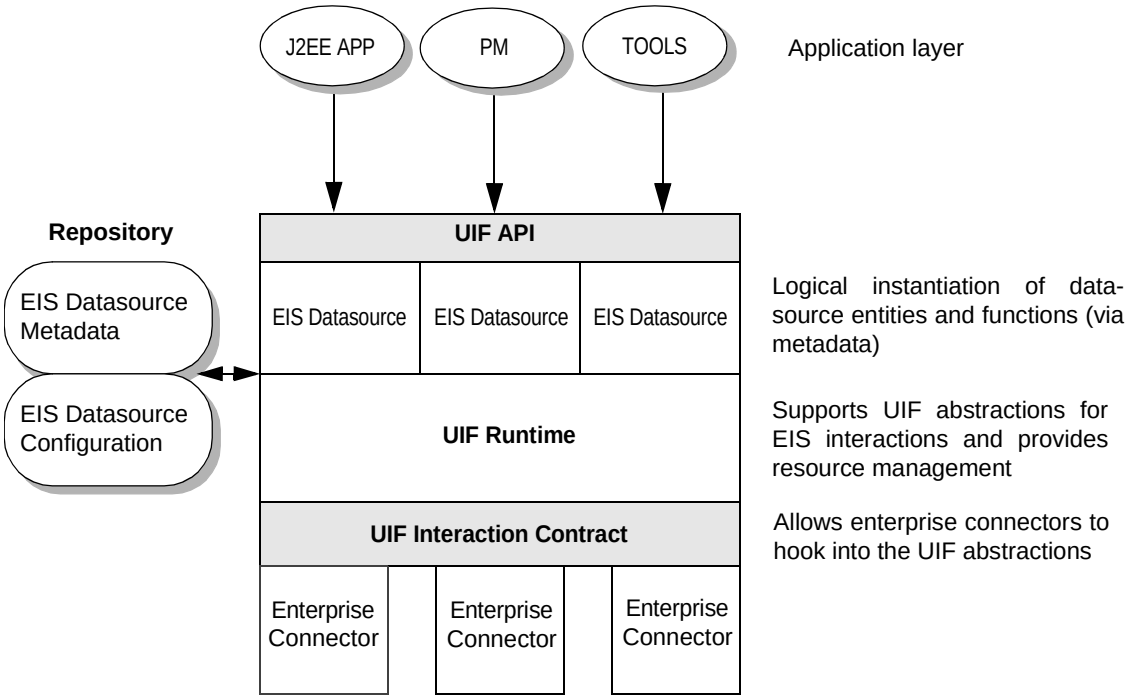


The UIF architecture includes a common set of APIs used to access data, UIF Runtime, UIF data repository, and an enterprise connector that implements the specific EIS data access services.

Application developers use the UIF APIs to uniformly access data. UIF runtime provides the connection management, thread management, resource pooling, user mapping, and the enterprise connector interaction services. A platform specific enterprise connector implements a UIF to enterprise connector contract, and provides the translation, data marshalling and unmarshalling between the UIF and the EIS. The UIF repository holds parameters and information about the EIS system, like the EIS definitions, data types, and business functions.

Figure 1-2 shows the UIF architecture layers, components and functions. In addition it also shows the application layer comprising of J2EE Applications or iPlanet Process Manager (iPM) applications or generic tools.

Figure 1-2 UIF Architecture Layers



A runtime snapshot of the UIF architecture comprises of the UIF APIs, UIF repository, UIF runtime, and the enterprise connector.

- UIF APIs – Java APIs for the application programmer to facilitate interactions with the enterprise connector.

- UIF Repository – A Lightweight Directory Access Protocol (LDAP) based repository which is used at runtime. The datasource administrator loads the UIF Repository at development time. The UIF Repository contains the datasource, function object and data object definitions. These definitions are used at runtime in order to instantiate the in-memory representation and interacts with the enterprise connector.
- UIF Runtime – This core piece enables all interactions between the application, repository and the enterprise connector. When the application requests to load a datasource for interaction, UIF runtime instantiates the in-memory representation of an EIS instance, which contains information on the location of the datasource and other access parameters. Further interactions on that datasource are facilitated by the UIF runtime, by loading the function object definitions from the repository and letting the application set the data object values and interacting with the enterprise connector by using various interaction models, such as Request / Response, Send / Receive etc. (UIF currently supports Request / Response model only), to execute the fully prepared function object.
- Enterprise Connector – The enterprise connector acts as a bridge between the UIF and the EIS. It adheres to the UIF interaction contract, and provides connectivity and interaction to the EIS. An enterprise connector, comes with a data mining and administration tools to mine the service definitions and interaction definitions into the UIF repository from the EIS.

  UIF in combination with a specific set of EIS enterprise connectors are used to build interactive web-based eBusiness applications and analytical applications for human resource management, financial, distribution, manufacturing, and supply chain management within an enterprise system.

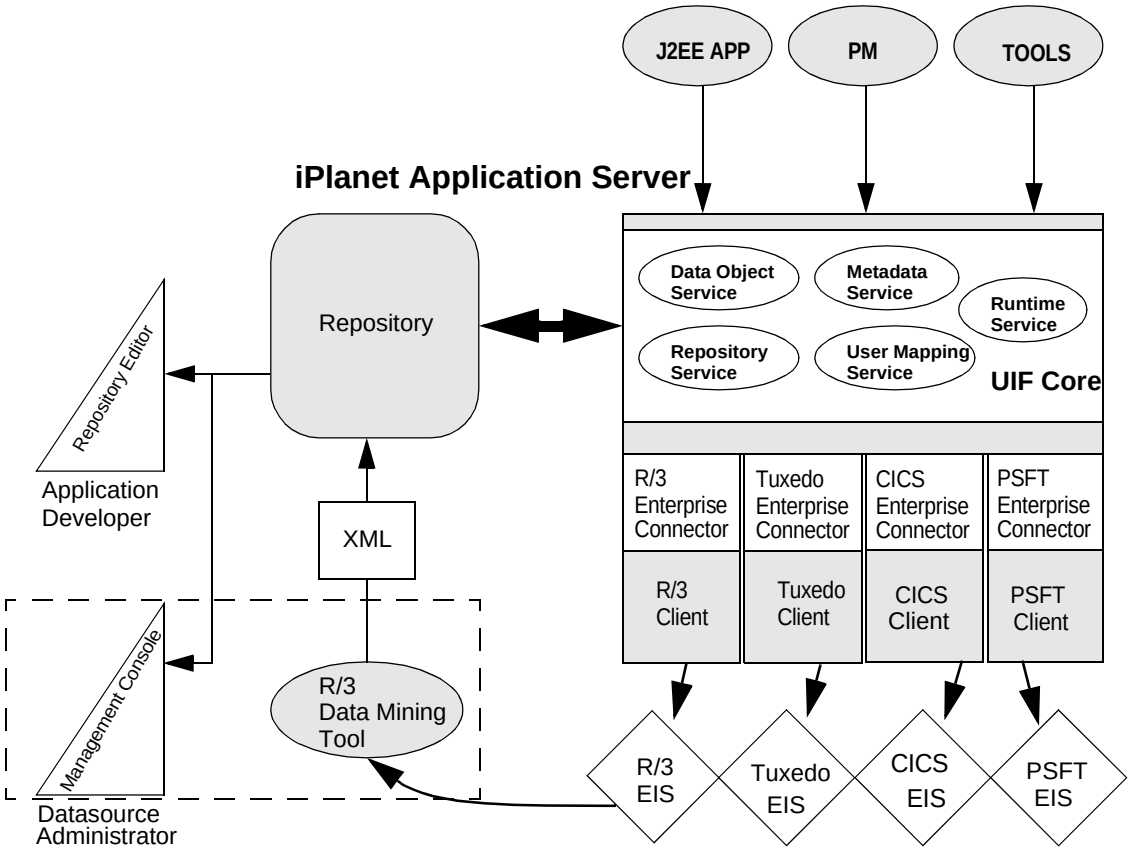
UIF Abstractions Overview

The UIF runtime is based on the following abstractions and services:

- UIF Internal Runtime Services
- Repository Schema and Definitions
- Enterprise Connector
- Execution Control Flow

Figure 1-3 shows a detailed view of the UIF components with an example of an iPlanet R / 3 Enterprise Connector.

Figure 1-3 UIF Components



UIF Internal Runtime Services

The UIF internal runtime services include the following services:

- Data Object Service
- Repository Service
- Metadata Service
- User Mapping Service
- Runtime Service

Data Object Service

The Data Object (DO) service implements a universal data representation object. A DO is a hierarchical data representation object, somewhat like a C/C++ structure. It can contain structures, arrays and lists, nested to arbitrary levels. DOs are self describing and support introspection. Programmatically, a DO is addressed using path names like *A.B[2].C*, and can iterate over DOs contents using iterators.

DOs pass all information and data across API boundaries between the application and UIF, and between UIF and the enterprise connector. The structural flexibility and introspectability of DOs allows the APIs between the three layers to be well defined and fixed.

Creation and destruction of DOs potentially has a high performance penalty due to their dynamic structure and flexibility. To reduce the potential penalty DOs are optimized via memory management and object reuse.

Repository Service

The Repository service models a persistent hierarchy of information that is like LDAP, but with a higher level of data contents. Persistence is provided on top of LDAP. The Repository service API is not exposed to the application. The Repository service users are mainly the UIF runtime and the enterprise connector management consoles.

Leaf nodes in the repository are either primitive key value nodes or complex data object type definition (dataTypes) nodes. Most data type nodes in the repository represent metadata mined from the EIS. They define types for parameter data blocks, configuration information blocks, and user mapping structures.

Data type nodes can be embedded and reused by other data type definitions. Also, a given data type can be used in multiple contexts where the field values for the data types are initialize in the different contexts. The fields are initialize via a template node which includes a data type reference and the initial field values for the data types.

The repository service maintains a read only cache of objects which are populated on demand. The cache does not refresh itself from LDAP once loaded. Therefore, changes to LDAP does not reflect in the repository service until the application server (KJS) is restarted. Note that the repository contains information which determines the enterprise connector behavior and cannot be used while in an inconsistent state. Changes to the repository are not updated until the application server is restarted.

The repository browser allows users to explore the repository contents. The tool has no editing features, however it does provide import, export, delete, and refresh actions on repository nodes. The repository contents are imported from the enterprise connector management console. An export function is also available via the repository browser which exports any subtree as an Extensible Markup Language (XML) document.

Although the repository service does not impose any specific organization (schema) on the repository contents, the UIF runtime requires the contents to be organized in a specific and well defined way. This organization is the UIF repository schema. Therefore, repository contents are not supposed to be arbitrarily modified by the user. Contents are modified as part of specific administration activities controlled by the management console, which makes appropriate programmatic use of the import and export functions. In addition, the UIF runtime service runs a consistency check on repository contents at startup.

Metadata Service

The Metadata service is used mainly by the UIF runtime service and is responsible for instantiating data objects from data type and template definitions in the repository. It also manages object reuse via data object freelists. The freelists are organized by data type and allow the administrator to set an upper boundary on memory usage, as well as other tuning parameters.

User Mapping Service

The User Mapping service has two separate user identity domains: web and EIS. An incoming request has a specific web user identity associated to it. The web identity needs to be mapped to an EIS user identity specific to the datasource.

The web user identity is set in the service provider configuration by the application. The enterprise connector maps the web user identity to the EIS user identity when the service provider is enabled (session state is created). The EIS user identity is maintained as part of the session state.

The User Mapping service maintains the user mapping tables. There are two distinct tables: EIS user identities and web-to-EIS mappings. The mapping service provides the enterprise connector with methods to determine an EIS user identity, given a web user identity as per the mapping tables. The EIS user table is loaded via a user mining function in the enterprise connector management console. The datasource administrator loads the user mapping table interactively via the management console (refer to your specific iPlanet Enterprise Connector *Administrator's Guide*).

Runtime Service

The Runtime service supports the EIS interaction abstraction and elements which support the interaction. The service currently supports only synchronous request and response interaction model. The runtime service includes core services for connection pooling and lifecycle management, thread management, conversation and session state management, and exception management. Most importantly, the service interprets UIF repository datasource information to create a UIF datasource object.

Two primary element objects of the UIF runtime are the serviceProvider and functionObject.

serviceProvider Object

A serviceProvider object represents a communication session with a specific datasource. The application creates a service provider by supplying a path to the definition in the repository. A serviceProvider's configuration (set by the datasource administrator) points to the EIS with the appropriate connection pooling and lifecycle management settings.

Underlying the serviceProvider is a connection object which is native to the enterprise connector and represents the EIS physical connection. The object is not readily understood by the UIF runtime. However, it is the UIF runtime's responsibility to manage the creation, destruction, and reuse (pooling and lifecycle management) of connection objects via callbacks into the enterprise connector. These settings are also part of the serviceProvider definition in the repository.

A serviceProvider has a session state object associated with it. This is used by the enterprise connector to keep session associated information, such as transaction identity and user identity. This state is not exposed to the application and the state object interacts with J2EE transaction and security models.

functionObject Object

A functionObject instance represents a specific business method available for execution on the datasource. The application creates a functionObject by supplying a path to the datasource definition in the repository. A functionObject definition is created by the data mining tool from metadata residing on the EIS.

A functionObject has one, but sometimes more enterprise connector specific operations defined on it. These operations execute enterprise connector specific interaction steps. For example, an R/3 functionObject may define a single **execBAPI** operation; whereas an MQseries functionObject may define three operations –

SEND/POLL/RECEIVE. When executed, an operation passes data to and from the enterprise connector via a data block. Since this data block represents inputs and outputs for a specific functionObject, data blocks of different function objects usually are of different data types.

In order to execute an operation, a functionObject requires a communication session. A communication session associates a serviceProvider with a functionObject before using the functionObject. A connector type may allow several functionObject interactions to execute over the same serviceProvider, in which case the interactions share session state (for example, transaction and user identity). A single functionObject interaction is also called a *conversation*, which occurs over a session. The conversation can also have a state which is shared by the operations which drive the conversation.

Repository Schema and Definitions

The UIF repository contains enterprise connector mined information on the EIS specific services, their input and output details, and other configuration information. This section explains the contents of the repository, how they are organized, and how UIF uses them.

The UIF runtime requires a specific organization in the repository. Currently no mechanism enforces this organization. In addition, the UIF runtime runs a schema validation check at startup.

Broadly, the organization has two sections: adapterTypes and dataSources. The adapterType section contains one subtree entry per enterprise connector. The entry contains common definitions which form the basis for defining datasources on that connector type. The adapterType entry is created when an enterprise connector is installed.

The dataSource section contains subtree entries, each of which is a logical UIF datasource. A datasource corresponds to a specific EIS and is supported by the corresponding connector type. Multiple datasources may be defined for a single connector type, if there are multiple EISs of the same type.

NOTE Datasources are defined depending on the EISs at a particular installation. The datasource definition activity is tightly controlled by the management console. There are two distinct user roles with datasources – the datasource administrator and the application developer. The datasource administrator is responsible for creating and administering datasource entries, while the application developer develops applications which access these datasources. A datasource definition is usually heavily preconfigured by the datasource administrator. Therefore, the application developer uses preconfigured function objects (much like using EJBs with deployment descriptor values already set).

Broadly, a datasource definition contains:

- serviceProvider definitions which represent a communication session to a specific EIS.
- Connection pool definitions.
- Data type definitions derived from metadata mined from the specific EIS.
- functionObject definitions which represent business methods available for execution on the specific EIS (for example, storedProc, prepared query, R/3 BAPI, PeopleSoft message, or CICS txn). These definitions are derived from metadata mined from the EIS.
- EIS user identity and user mapping tables.
- Connector type specific configuration information which controls the enterprise connector's interaction with the specific EIS.

Please refer to Chapter 3, “Repository” for details.

Enterprise Connector

The enterprise connector's main responsibility is to act as a bridge between UIF and EIS. It manages the communication with the EIS, and data marshalling and unmarshalling between data objects and EIS formats. The UIF provides an enterprise connector side contract consisting of several interfaces which the enterprise connector implements in order to integrate with UIF. The EIS specific implementation of this contract is the essence of an enterprise connector. The UIF and enterprise connector contract interfaces are broadly separated into the following three categories:

- Connection pooling callbacks – used for creating, matching and reserving, returning, and destroying connections.
- Communication model callbacks – used for creating and destroying conversation and session state objects.
- Execution callbacks – used for executing operations. All data marshalling and unmarshalling is done here.

The enterprise connector also has an exception throwing mechanism. The enterprise connector can throw an exception which propagates upwards through the UIF layer and emerges in the Java application as a Java exception. For enterprise connector specific exception information refer to your specific iPlanet Enterprise Connector *Developer's Guide.*

Management Console

The management console is used to view and change an EIS configuration in the UIF Repository. Each enterprise connector has a specific management console. However, all management consoles allow an administrator to perform the following tasks:

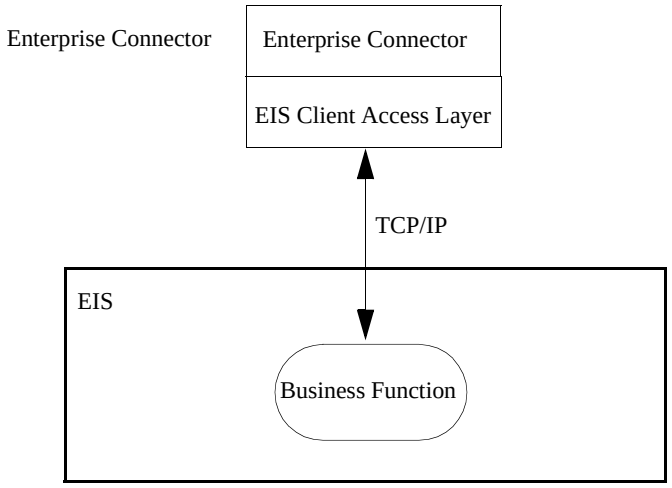
- Create a new datasource
- Configure a datasource, for example, setting connectivity and I18N parameters to the specific EIS instance
- Configure pooling parameters
- Mine and load function objects into the respository
- Configure user mapping
- Monitor and manage EIS specific functions, for example, a CICS *relay* program

For enterprise connector specific management console information refer to your specific iPlanet Enterprise Connector *Administrator's Guide.*

Enterprise Connector-to-EIS Architecture

Figure 1-4 shows the components overview between the Enterprise Connector and the EIS.

Figure 1-4 Enterprise Connector to EIS



The enterprise connector lives as part of the application server during runtime and gets replicated as the iPlanet Application Server replicates.

The enterprise connector uses an EIS specific client access layer to provide the connection and interaction. The enterprise connector uses an EIS specific client access layer to supply the connection. The connection protocol between the client access layer and EIS, is EIS specific (usually Transmission Control Protocol / Internet Protocol (TCP / IP), but in some cases other protocols are used, for example CICS uses SNA LU 6.2). UIF maintains a connection pool for each EIS and each connection is reused for multiple application requests.

Depending on the specific EIS, establishing communication between the enterprise connector and EIS may require installing an extra components on the EIS. For example, CICS requires a *relay* program. For more information about pooling, see Chapter 4, “Pooling Concepts”.

Execution Control Flow

The application developer uses the repository browser and decides to execute a specific function object under a specific datasource. General usage and control flow are outlined below (not all steps and callbacks are described):

1. Creates the service provider by specifying a repository path
2. Sets the service provider configuration data object fields (for example, web userid)

3. Enables the service provider
4. Creates the function object by specifying a repository path
5. Associates the service provider with the function object
6. Sets the inputs on the data block of the function object
7. Executes the operation on the function object
8. Disables the service provider

   Refer to the Chapter 5, “Programming Model” for further details.

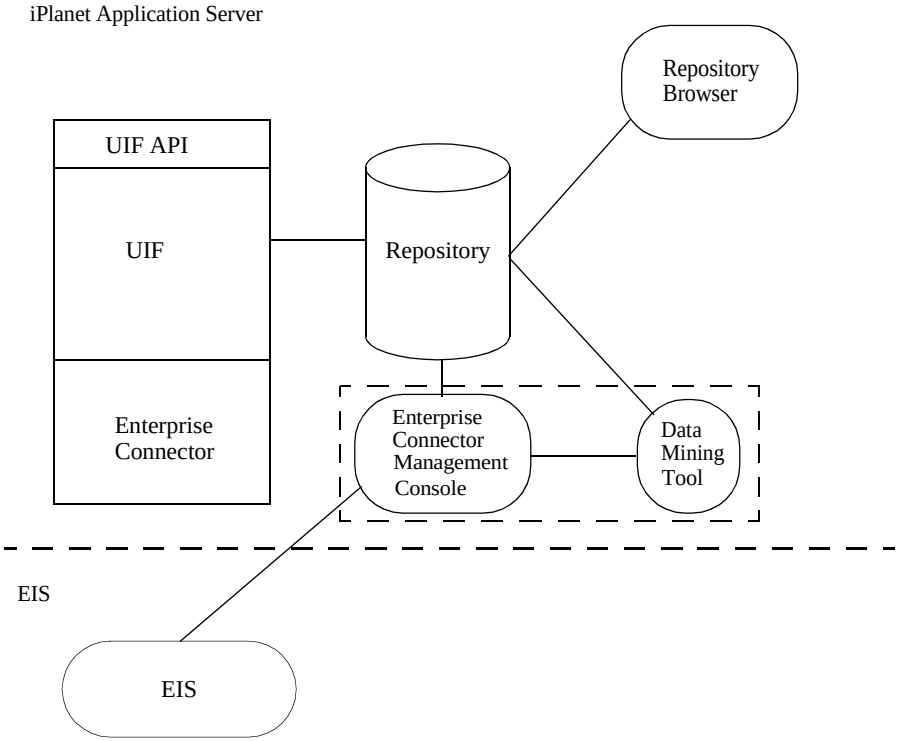
UIF Tools

The UIF tools consist of the Repository Browser, the Enterprise Connector specific management console, and Data Mining tool, and are used as follows:

- The Repository Browser – is used to view the repository contents, and to import and export EIS metadata. It allows developers to discover EIS business functions available for execution.
- The Management Console – is used to browse or change the enterprise connector configuration, and to monitor EIS status. Each enterprise connector has an enterprise connector specific management console.
- The Data Mining Tool (is part of the management console and is not a separate tool) – is used to determine the EIS's available transactions, and to load its meta information into the repository as function objects available for execution by a J2EE application.

Figure 1-5 shows the tools used to configure UIF.

Figure 1-5 iPlanet Application Server to EIS Connection



UIF API

UIF provides a Java API to enable a J2EE application developer to uniformly interact with the disparate datasources. The API gives fine grained control over the interaction with the EIS. Please refer to the Chapter 7, “API Reference” for details.

UIF Application Development Model

After installing UIF and the enterprise connector on the iPlanet Application Server and the EIS, its time to develop a J2EE Component to invoke an EIS-based transaction. The following steps are recommended:

1. Run the data mining tool from the management console to create repository entries for function objects which need to be execute. For more information on the management console, refer to your specific iPlanet Enterprise Connector *Administrator’s Guide*.
2. Import the EIS services as function objects using the management console.
3. Select the datasource in the repository browser to view the:
   a. service provider definition for the configuration attributes needed to establish a connection
   b. function object definition to determine:

      Use these parameters for the UIF API methods to call.

      I. what (if any) property set attributes are required
      II. what inputs are expected in the data block
      III. what outputs are defined in the data block

   For more information about using the repository browser, see “Using the UIF Repository Browser” on page 70.
4. Write the J2EE components which use the UIF API. For an overview of the UIF API and sample code, see Chapter 5, “Programming Model”. For more details about the UIF API, see Chapter 7, “API Reference”.

5. After writing and testing a Servlet, JSP, or EJB, use the iPlanet Application Server Deployment Manager to deploy the application. Explicitly deploy the repository contents by exporting its contents to XML and importing the XML file on the deployment machine. For more information about exporting and importing XML files, see “Using the UIF Repository Browser” on page 70. For information about iPlanet Application Server deployment, see the *iPlanet Application Server Deployment Guide*.

Its now time to execute a J2EE component within your application. Refer to the Chapter 5, “Programming Model” for further details.

Chapter 2

Installation

This chapter describes how to install and configure UIF on the iPlanet Application Server.

The following topics are included in this chapter:

- Summary of Installation Tasks
- Checking Your Package
- Checking Hardware and Software Requirements
- Preparing to Install
- Installing UIF
- Uninstalling UIF
- Troubleshooting Your Installation
- Upgrade or Reinstall Issues
- Cluster Installations

Summary of Installation Tasks

To install UIF, complete the following tasks:

- Complete pre-installation tasks as needed, as described in “Preparing to Install,” on page 39.
- Install UIF according to the instructions for your target platform in “Installing UIF,” on page 39.

Checking Your Package

The UIF package is included with your iPlanet enterprise connector which includes the following items:

- One iPlanet Application Server Unified Integration Framework Install CD-ROM containing installation software, on-line documentation, and README.txt file.
- One iPlanet Application Server Enterprise Connector Install CD-ROM contain installation software, online documentation, and README.txt file.
- One *iPlanet Application Server Enterprise Connector Administrator's Guide*.
- One *iPlanet Application Server Enterprise Connector Developer's Guide*.
- One *iPlanet Unified Integration Framework Developer's Guide*.

Checking Hardware and Software Requirements

The following table lists the hardware and software requirements for installing and running UIF.

Component	Requirement
Computer and Operating System	One of the following systems: • Intel Pentium microprocessor running Microsoft Windows NT version 4.0 with Service Pack 5 • Sun SPARC running Solaris 2.6 • Sun SPARC running Solaris 8
Memory	Per CPU: 128 MB minimum; 256 MB recommended
Available disk space	Total disk space: 200 MB on NT, 400 MB on Solaris, 800 MB on Solaris if installing the iPlanet Application Server 6.0
CD-ROM drive	CD-ROM drive
Network software	TCP / IP
Browser software	One of the following web browsers: • Netscape Communicator 4.5 • Microsoft Internet Explorer 4.0 (NT only)
Other software	• iPlanet Application Server 6.0 SP1 • The iPlanet Application Server must be installed prior to installing UIF.

Preparing to Install

Before you install UIF complete the following steps:

1. Verify that the target system is properly configured and has all the prerequisite software installed and running, as described in “Checking Hardware and Software Requirements,” on page 38.
2. Before you install UIF, stop the iPlanet Application Server if it is currently running. The installation program may not be able to overwrite files that are currently in use.
3. Read the UIF *Release Notes* for any last minute updates.

Installing UIF

The UIF installation procedure adds the following components to your system:

- UIF software
- Documentation

This section describes how to install UIF on the iPlanet Application Server. It contains the following topics:

- Installing on Windows NT
- Installing on Unix

Installing on Windows NT

The iPlanet Application Server 6.0 SP1 is necessary software that must be installed before starting the UIF installation.

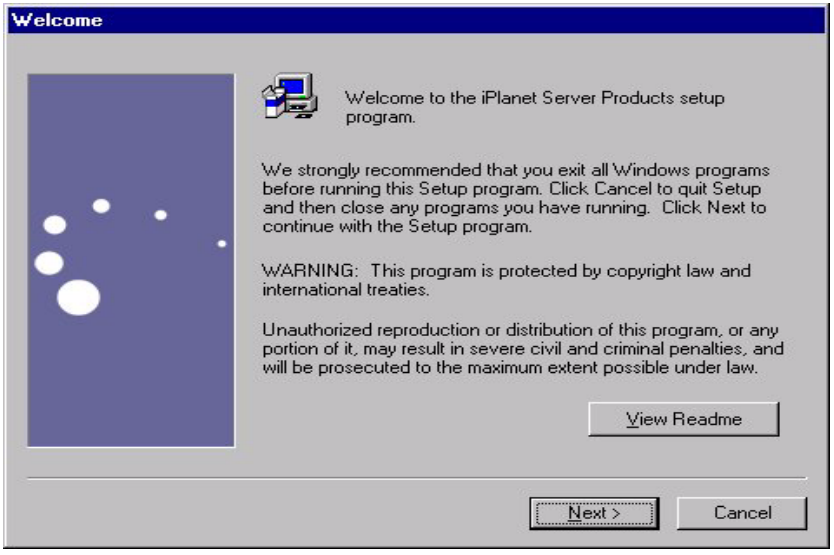
You must be logged on to Windows NT as a user with administrator privileges to install UIF.

To install UIF complete the following steps:

1. Read the *README.txt* file on the Installation CD-ROM.
2. Copy ***NT/insbsp.zip*** from the Installation CD-ROM into a temporary local directory.
3. Unzip ***insbsp.zip***, then run ***setup.exe***.

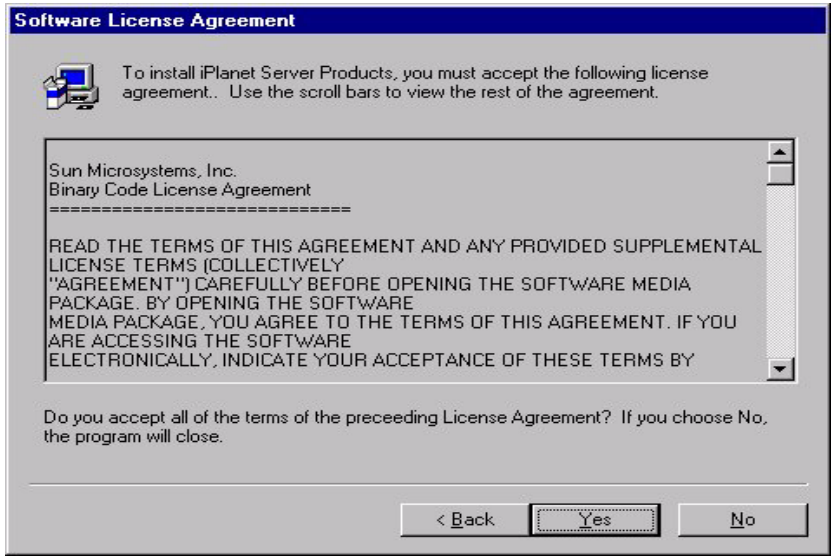
4. Read the Welcome message and then click on the Next button.

Figure 2-1 Welcome Window



5. The next window that appears is the Software License Agreement. Use the scroll bar or the Page Down key to read the License Agreement. Click on the Yes button if you agree.

Figure 2-2 Software License Agreement Window



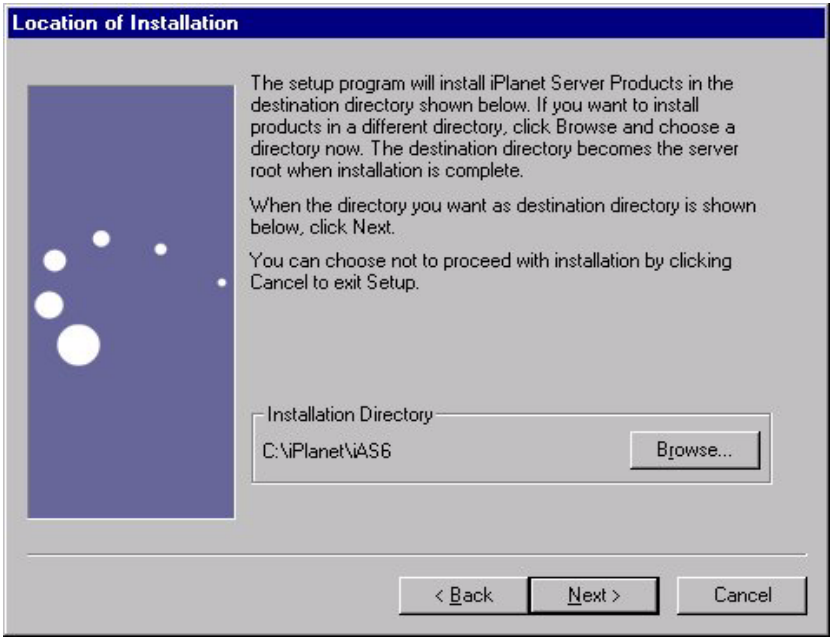
6. The next window that appears is the Location of Installation. Choose your installation directory by clicking on the Browse button and navigate to the desired directory.

   Click the Next button after making your selection.

NOTE UIF must be installed in the same directory as the iPlanet Application Server.

If installer detects that iPlanet Application Server software is not installed, a message is displayed indicating that you cannot continue with installation. Click OK to exit Setup.

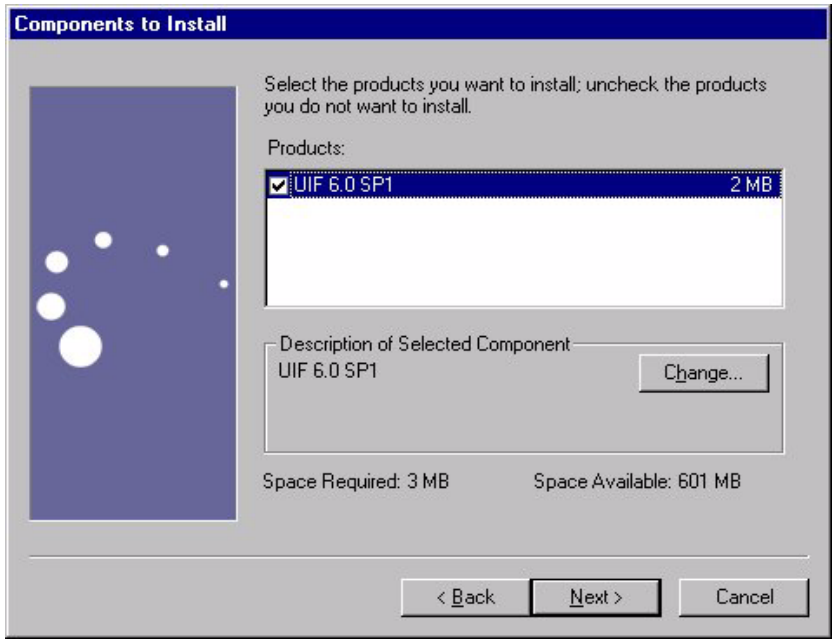
Figure 2-3 Location of Installation Window



7. The Components to Install window appears. Select the UIF 6.0 SP1 checkbox or click on the Change button to select the desired components to install.

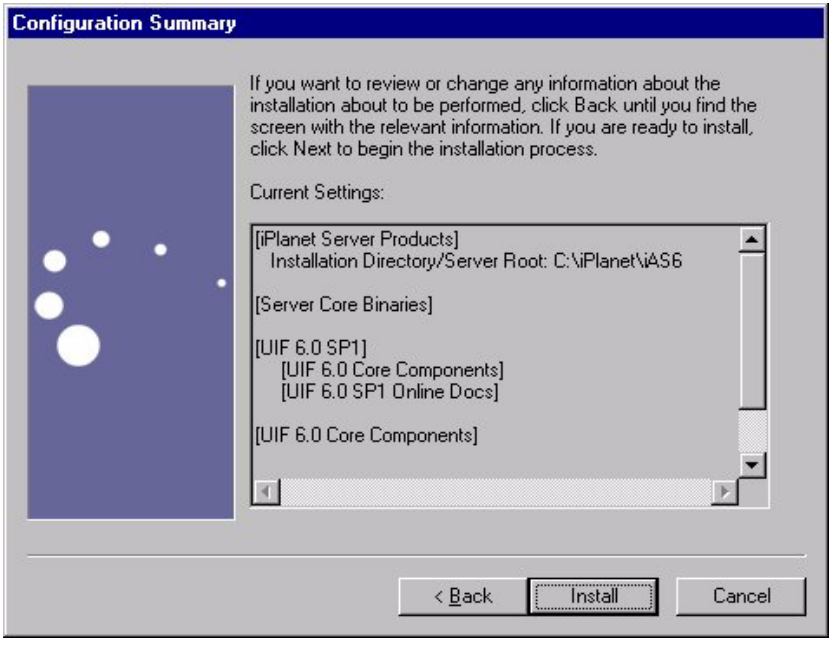
   Click the Next button after making your selection.

Figure 2-4 Components to Install Window



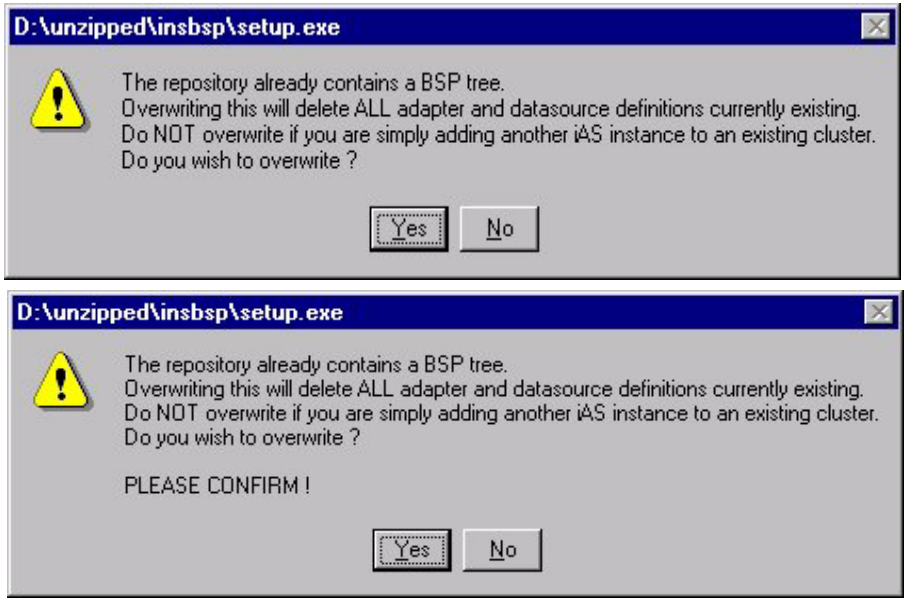
8. Review the Configuration Summary. If it is correct, press the Install button.

Figure 2-5 Configuration Summary Window



9. The installer displays these warning messages if you have already installed UIF, and you are reinstalling or upgrading UIF.

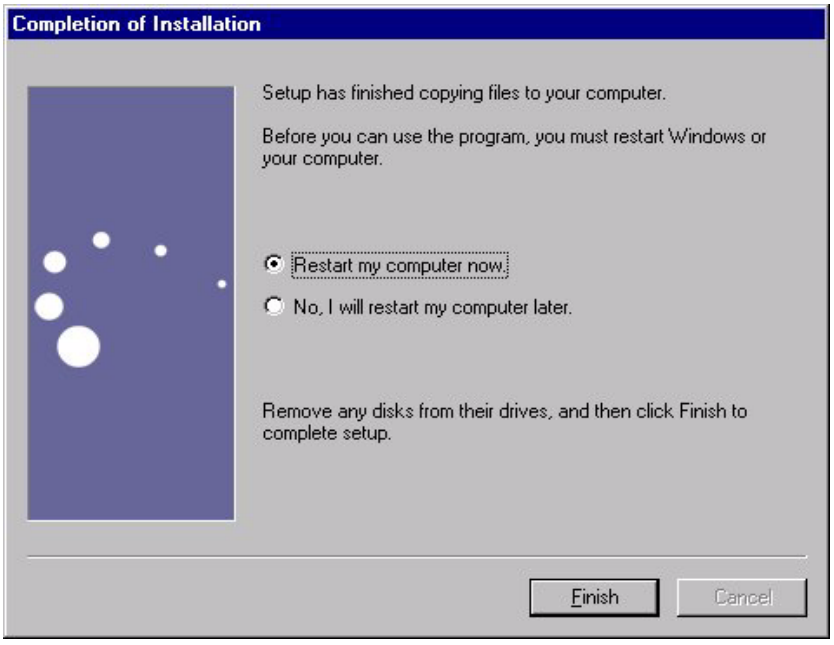
Figure 2-6 BSP Tree Overwrite Warning Messages



10. When all files are installed, the Completion of Installation window displays. Click on Finish.

    UIF has installed correctly and the system must now be rebooted to restart all iPlanet Application Server engines.

Figure 2-7 Completion of Installation Window



Installing on Unix

The iPlanet Application Server 6.0 SP1 is necessary software that must be installed before starting the UIF installation.

The following procedure describes how to install UIF on Unix:

1. Insert the UIF Install CD-ROM into the CD-ROM drive.
2. Mount the CD-ROM on, for example `/cdrom/cdrom0`.
3. Copy ***Solaris/insbsp.tar*** from the Installed CD-ROM into a temporary directory.
4. Untar ***insbsp.tar***, then run `./setup` command.

5. Follow the instructions of the installation program. During installation, press:

   CTRL+B to back up to the previous screen within an installation section.

   CTRL+C if you need to exit the installation. This results in an incomplete installation. If you want to install again, you must run the installation program from the beginning.

6. The following welcome screen appears. Type Yes to continue.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
---------------------------------------------------------------

Welcome to the iPlanet Server Products installation program
This program will install iPlanet Server Products and the
iPlanet Console on your computer.

It is recommended that you have "root" privilege to install the
software.

Tips for using the installation program:
  - Press "Enter" to choose the default and go to the next screen
  - Type "Control-B" to go back to the previous screen
  - Type "Control-C" to cancel the installation program
  - You can enter multiple items using commas to separate them.
    For example: 1, 2, 3

Would you like to continue with installation? [Yes]:
```

7. The following license agreement screen appears. If you agree with the terms of the license agreement, type Yes to continue.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
---------------------------------------------------------------

BY INSTALLING THIS SOFTWARE YOU ARE CONSENTING TO BE BOUND BY
AND ARE BECOMING A PARTY TO THE AGREEMENT FOUND IN THE
LICENSE.TXT FILE. IF YOU DO NOT AGREE TO ALL OF THE TERMS
OF THIS AGREEMENT, PLEASE DO NOT INSTALL OR USE THIS SOFTWARE.

Do you agree to the license terms? [No]:
```

8. Press enter or specify 1 to select “iPlanet Servers” as the item to install.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
---------------------------------------------------------------
Select the items you would like to install:

   1. iPlanet Servers
       Installs iPlanet Servers with the integrated iPlanet
      Console onto your computer.

   2. iPlanet Console
       Installs iPlanet Console
       as a stand-alone Java application on your computer.

To accept the default shown in brackets, press the Enter key.

Select the component you want to install [1]:
```

9. Specify the iPlanet Application Server installation directory. This is the iAS root directory.

UIF must be installed in the same directory as the iPlanet Application Server.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
---------------------------------------------------------------

This program will extract the server files and install them
into a directory you specify. That directory is called the
server root in the product documentation and will contain
the server programs, the Administration Server, and the server
configuration files.

To accept the default shown in brackets, press the Enter key.

Install location [/iPlanet/iAS6]:
```

10. The following “iPlanet Server Products components” screen appears. Select “UIF6.0SP1” to install.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
---------------------------------------------------------------

iPlanet Server Products components:

Components with a number in () contain additional subcomponents
which you can select using subsequent screens.

    1. UIF6.0SP1 (1)

Specify the components you wish to install [All]:
```

11. The following “UIF6.0SP1 components” screen appears. Press enter to install all the components.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
-----------------------------------------------------------------

UIF6.0SP1 components:

Components with a number in () contain additional subcomponents
which you can select using subsequent screens.

    1. UIF core components
    2. Online Docs

Specify the components you wish to install [1, 2]:
```

12. Specify fully qualified domain name of the computer on which you are installing and press the Enter key.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
-----------------------------------------------------------------

Enter the fully qualified domain name of the computer
on which you’re installing server software. Using the form
<hostname>.<domainname>
Example: eros.airius.com.

To accept the default shown in brackets, press the Enter key.

Computer name [horse.red.iplanet.com]:
```

13. Specify the Unix user and group name under which UIF runs.

You should have already set up this user and group name prior to running this installation.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
-----------------------------------------------------------

Choose a Unix user and group to represent the iPlanet server
in the user directory.  The iPlanet server will run as this user.
It is recommended that this user should have no privileges
in the computer network system.  The Administration Server
will give this group some permissions in the server root
to perform server-specific operations.

If you have not yet created a user and group for the iPlanet
server,create this user and group using your native UNIX
system utilities.

To accept the default shown in brackets, press the Return key.
System User [nobody]: root
System Group [nobody] : other
```

14. Press Enter key to continue.

```
iPlanet E-Commerce Solutions, Sun-Netscape Alliance
Unified Integration Framework Installation/Uninstallation
-----------------------------------------------------------------

Finished with gathering of information.

Next, the Unified Integration Framework files will be extracted.
After that, the configuration and registration based on your
responses will begin.

Please hit ENTER to continue:
```

15. The installer displays these warning messages if you have already installed UIF, and you are reinstalling or upgrading UIF.

```
iPlanet E-Commerce Solutions, Sun-Netscape Alliance
Unified Integration Framework Installation/Uninstallation
-------------------------------------------------------------
The repository already contains a BSP tree.
Overwriting this will delete *all* adapter and datasource
definitions currently existing.
Do *not* overwrite if you are simply adding another iAS instance
to an existing cluster.
Do you wish to overwrite ?

Confirm? [No]: y
```

```
iPlanet E-Commerce Solutions, Sun-Netscape Alliance
Unified Integration Framework Installation/Uninstallation
-------------------------------------------------------------
The repository already contains a BSP tree.
Overwriting this will delete *all* adapter and datasource
definitions currently existing.
Do *not* overwrite if you are simply adding another iAS instance
to an existing cluster.
Do you wish to overwrite ?

PLEASE RE-CONFIRM !

Confirm? [No]: y
```

16. When all files are installed, the Completion of Installation screen appears. Press Return to continue. The system displays the following message:

```
GO to <iplanet installation directory> & type startconsole to
begin managing your servers.
```

Uninstalling UIF

This section describes how to uninstall UIF from your iPlanet Application Server. It contains the following topics:

- Uninstalling on Windows NT
- Uninstalling on Unix

Uninstalling on Windows NT

Perform the following steps to uninstall UIF from a Windows NT:

1. Choose Uninstaller from Windows NT Start menu's Programs - iPlanet Application Server 6.0 entry. The following iPlanet Uninstall window is displayed:

Figure 2-8 iPlanet Uninstall Window

2. Deselect all components except UIF 6.0 SP1, then click Uninstall button.

   While deselecting the components for UIF, the uninstaller checks for dependencies between components.

   Deselect the components in the following order:

   - Netscape Server Products Core Components
   - Administration Services
   - Netscape Directory Suite
   - iPlanet Application Server 6.0

   Deselect the subcomponents for iPlanet Application Server 6.0 in the following order:

   - Application Server 6.0 Core Binaries
   - Web Connector plug-in Component
   - Administrator Tool
   - Deployment Tool
   - Core Server Component

UIF has been removed.

If you do not follow the above mentioned order of deselection, alert messages are displayed and you may not be able to deselect the desired option.

Uninstalling on Unix

Perform the following steps to uninstall UIF from a Unix:

1. Go to iPlanet Application Server installation directory and type `./uninstall`. The following screen is displayed.

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
---------------------------------------------------------------

The following are the iPlanet
components currently installed on your machine:

Components with a number in () contain additional subcomponents
which you can select using subsequent screens.

    1. iPlanet Server Products Core Components (3)
    2. iPlanet Directory Suite (2)
    3. Administration Services (2)
    4. iPlanet Application Server Suite (4)
    5. UIF 6.0 SP1(2)

Select the components you wish to uninstall (default: all) [All]:
```

2. Select the UIF6.0SP1 entry. The following screen is displayed:

```
iPlanet E-Commerce Solutions, A Sun-Netscape Alliance
iPlanet Server Products Installation/Uninstallation
-------------------------------------------------------------

The following are the UIF 6.0 SP1
components currently installed on your machine:

Components with a number in () contain additional subcomponents
which you can select using subsequent screens.

    1. UIF core components
    2. Online Docs

Specify the components you wish to uninstall [1, 2]:
```

3. Select all subcomponents.

UIF has been removed.

Troubleshooting Your Installation

If a problem occurs during installation:

- Review the system configuration as described in “Checking Hardware and Software Requirements,” on page 38.
- Verify that iPlanet Application Server is installed properly.
- Verify that iPlanet Unified Integration Framework is installed properly.
- Verify that you completed the installation steps correctly, as described in “Installing UIF,” on page 39.
- Check the logs of KAS, KXS, and KJS engines for additional information.

Review the following list for possible problems and suggested action(s):

Installer unable to detect iPlanet Application Server components	Check iPlanet Application Server software has been installed prior to installing UIF
Insufficient memory	Close other open programs
Network connection failed	See your network administrator
Insufficient disk space	Remove unnecessary files from the target volume

Upgrade or Reinstall Issues

If you are upgrading an earlier version or if you are reinstalling the current version of UIF, the installation procedure simply overwrites the installed files. You do not need to uninstall them first. Please refer to your *iPlanet Application Server Enterprise Connector Administrator's Guide* for details on backing up your earlier repository contents and upgrade issues.

NOTE In all cases, to avoid problems with overwriting files that might be in use, stop the iPlanet Application Server before installing software. Also, backup the system prior to installation including the UIF repository content. UIF repository contents can be backed up by using the UIF Repository Browser. Use UIF Repository Browser export functionality to export the Adapter type and Data Source definitions to XML files.

Cluster Installations

A cluster is a group of iPlanet Application Servers connected in the same network that synchronize data. Data is shared by all servers in the cluster which is stored in an iPlanet Directory Server. In most cases, each server in the cluster share common Directory Server.

UIF and Enterprise Connector Clusters

In a server cluster, all iPlanet Application Servers, UIF, and enterprise connector instances share the same LDAP instance and configuration node.

- Although the UIF repository is shared across all iPlanet Application Servers in a cluster, each server in the cluster may or may not have an enterprise connector installed. If a server does not have an enterprise connector installed, it sees no datasource for the missing connector type. All servers with a given enterprise connector installed see the same set of datasources for that type.
- The UIF Repository Browser is used from any iPlanet Application Servers, UIF, and enterprise connector instances to manage the cluster's LDAP BSP (UIF) node.
- The Enterprise Connector Management Console is used to configure the enterprise connector, User Management and Data Mining for all cluster instances.

For details about webless and cluster installations refer to the *iPlanet Application Server Installation Guide* for details.

Chapter 3

Repository

This chapter describes the UIF Repository and the UIF Repository Browser. For a description of the repository schematics and abstractions, refer to the Chapter 1, “Concepts”. For a detailed description on the EIS specific content, see your specific iPlanet Enterprise Connector *Developer’s* and *Administrator’s Guides*.

This chapter contains the following sections:

- UIF Repository
- About the UIF Repository Browser
- Using the UIF Repository Browser
- Troubleshooting Tips

UIF Repository

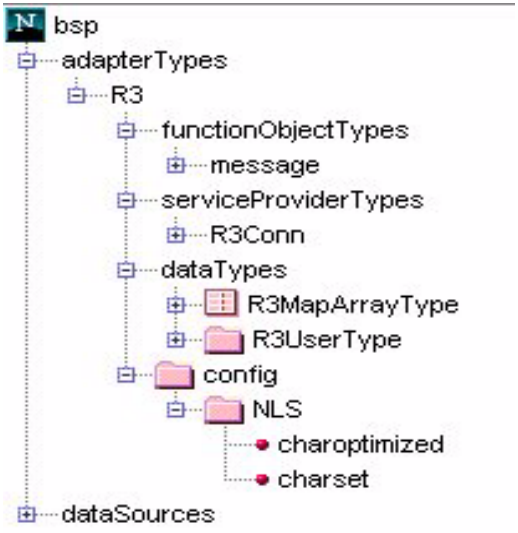
UIF stores information in the repository which defines the EIS business functions available to an iPlanet Application Server Servlet, JSP, or EJB. Repository data includes metadata definitions that are one of two types, as follows:

- Connector Types (*adapterTypes*) – Defines the enterprise connector specific characteristics including supported function object types and logical connection types (service provider types). This forms the basis for defining the datasources of this connector type.
- Datasources – Defines a logical datasource and represents a specific EIS instance (system). A datasource definition specifies the business functions available on a specific EIS. The datasource defines the actual functions available (function objects), the supported connections (service providers), and other information.

Connector Types

The connector type (*adapterTypes*) definitions correspond to each installed enterprise connector on an iPlanet Application Server. They form the basis of a datasource definition in the repository. A connector type contains one or more function object types, a service provider type, and configuration information. A type definition is created when the enterprise connector is installed. Figure 3-1 shows a typical R/3 (R3) connector type.

Figure 3-1 Connector Types

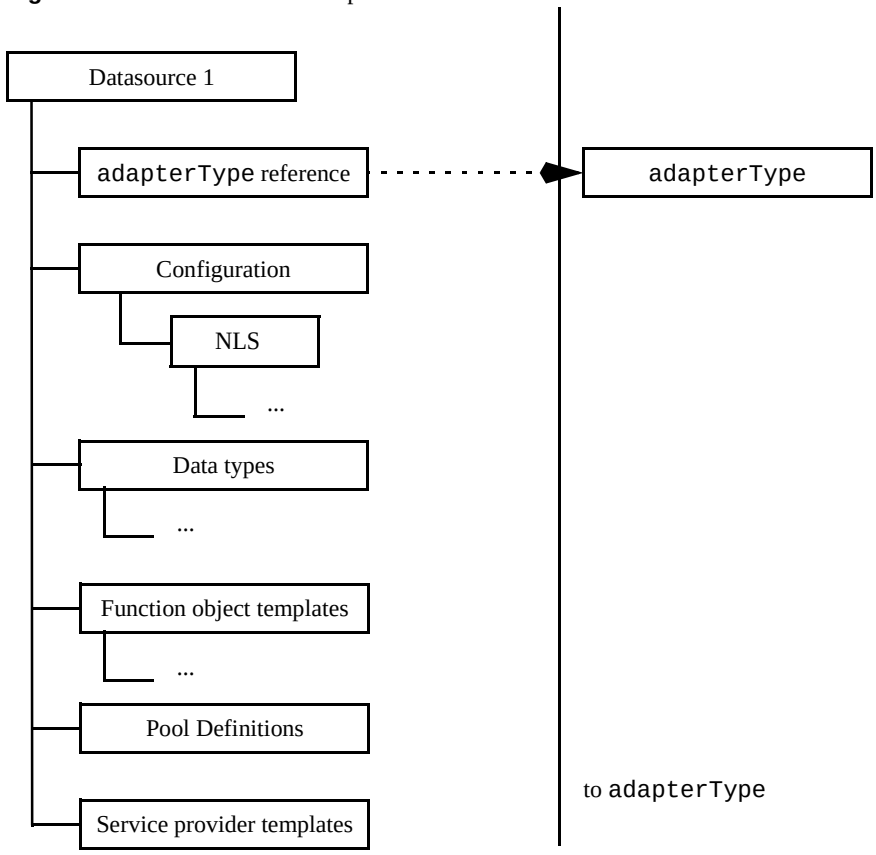


Datasources

A datasource represents a single EIS instance and defines the business functions available for execution. For example, an organization might have two systems or subsystems, one for personnel (human resources) and one for accounting. You can choose to represent each as a different datasource even if both run on the same connector type.

An application developer writing a J2EE (Servlet, JSP, or EJB) component uses the datasource definition in the respository to determine the function objects and service provider available on a particular datasource system. Figure 3-2 shows a datasource's basic components.

Figure 3-2 Datasource Components



Datasources

A datasource consists of the datasource configuration information, data type definitions, service provider definitions, function object definitions, pool definitions, and user mapping information. The following sections describe these components in more detail.

Datasource Configuration Information

The datasource configuration information is the EIS specific configuration settings which the enterprise connector interprets and uses to control its interaction with a specific EIS. In addition, each datasource specifies the character set and passes the data back and forth through the enterprise connector. For a detailed description of the default character set, available EIS character sets and how to specify them, see your specific iPlanet Enterprise Connector *Developer's* and *Administrator's Guides.*

The values of the datasource configuration settings control the enterprise connector's interaction with the EIS instance corresponding to this datasource.

Data Type Definitions

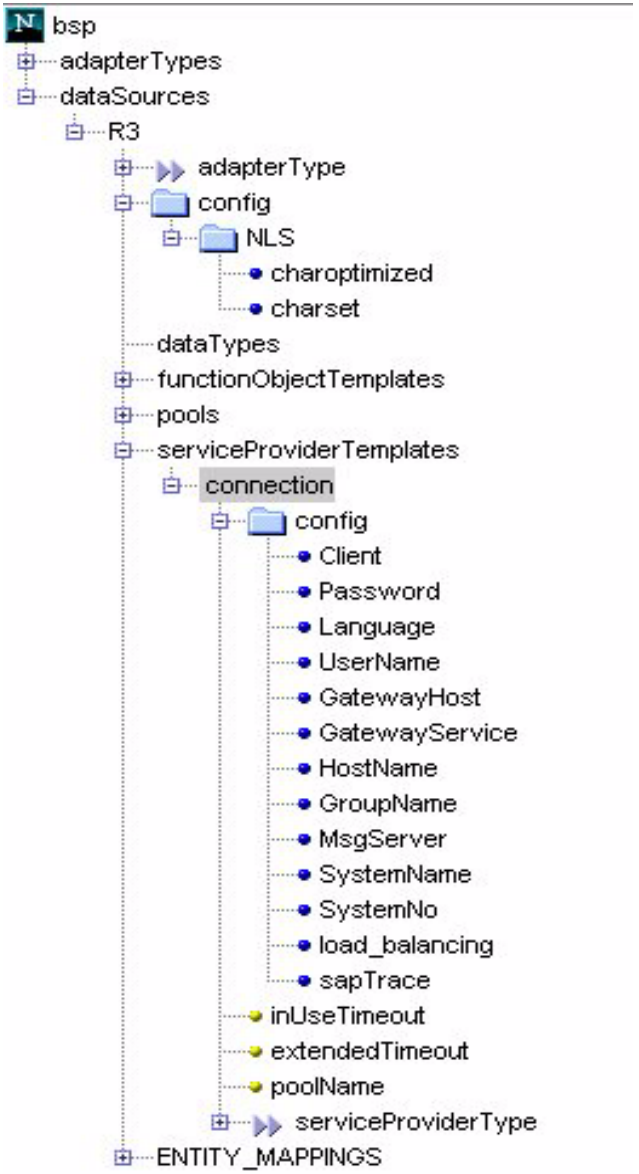
Data type definitions are used by the metadata service to instantiate data objects. All information between the application and the enterprise connector is passed using data objects. The definitions represent structured EIS data types and are mined from the EIS metadata. Each specific EIS may have its own set of data types, which is why they are associated with the datasource and not the connector type. Data types may represent parameter blocks, messages, events, etc. – depending on the specific EIS. The enterprise connector is responsible for marshalling and unmarshalling data objects, to and from the specific EIS formats.

Service Provider Definitions

A service provider represents a connection to an EIS system. It contains configuration information required to create a connection. For example, it may define host, port, username, and password. A service provider type may also specify additional information required to open or manage a connection.

The definition includes default EIS specific configuration settings, whether the connection is pooled or not, which pool is assigned, duration of the service provider connection binding, binding timeouts, and thread handoff settings. Figure 3-3 shows a typical service provider connection for R/3.

Figure 3-3 Service Provider Connections



Function Object Definitions

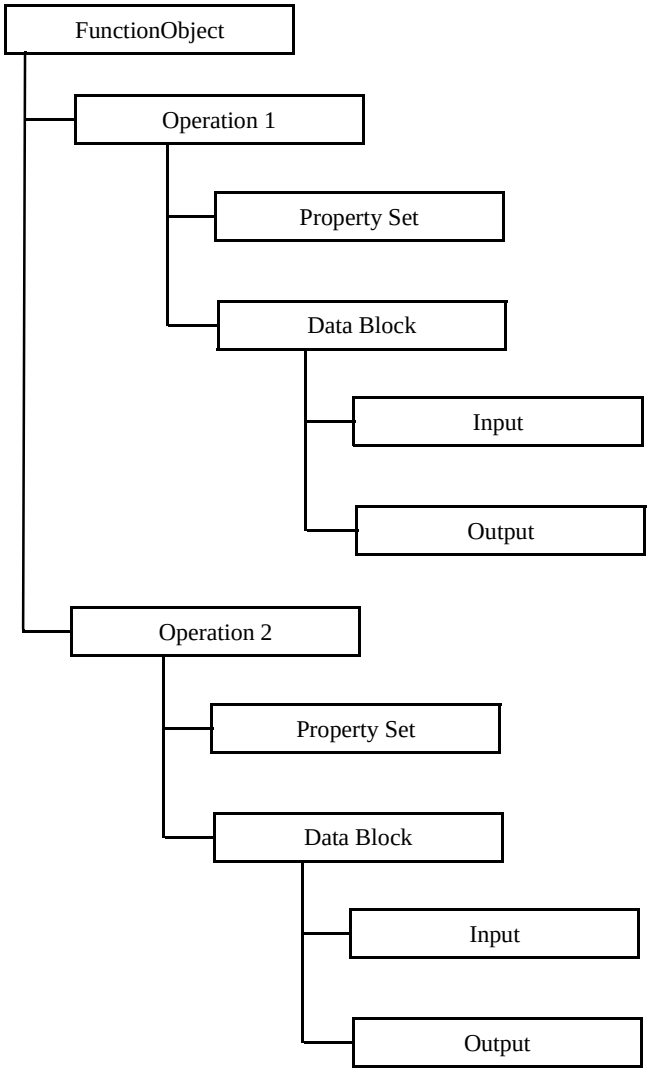
Function objects provide logical definitions of business functions available on the EIS datasource. A specific business function object represents a specific EIS type, for example:

- BAPI – R/3
- RFC – R/3
- psft message – PeopleSoft
- CICS txn – CICS
- and so on, depending on the specific EIS

Each EIS may have its own set of business functions which is why they are associated with the datasource and not the connector type.

Function object creation in the repository uses an EIS specific data mining tool supplied with each enterprise connector. Figure 3-4 shows a typical function object.

Figure 3-4 Function Object



Operations

A function object itself exposes one or more named *operations*. For example, a message function object may expose three operations – SEND, POLL, RECEIVE. However, most systems only need one operation (default). Each operation has an associated propertySet and dataBlock.

Property Sets

A property set is a data object used to target a specific EIS business function and describe the various operational parameters. For example a property set may specify: BAPI name, peopleSoft message identifier, program name, queue name, EIS host / port / URL, etc. The enterprise connector uses the property set contents to identify a specific EIS business function.

Data Blocks

A data block is a data object used to encapsulate all inputs and outputs between the application and the EIS. This object can have a default or initial value(s). The enterprise connector is responsible for marshalling and unmarshalling data blocks, to and from specific EIS formats. Therefore, a function object targeting a specific business method has its own specific associated data block type.

A data block describes each business logic program's input and output parameters. In UIF, each data block has two child nodes – input and output.

Function Object Template

A function object template is a function object instance. A function object template contains a data block definition and a property set whose values define access to a specific EIS transaction. A data block represents the operation's parameter block and holds the input and output information for the operation's execution.

dosearch – is a named function object, representing a specific EIS business function which accepts an account number as input and returns activity details as output. The application points to the named function object and creates a runtime function object instance. When the execute method is called on this object, the enterprise connector executes the EIS business function.

Pool Definitions

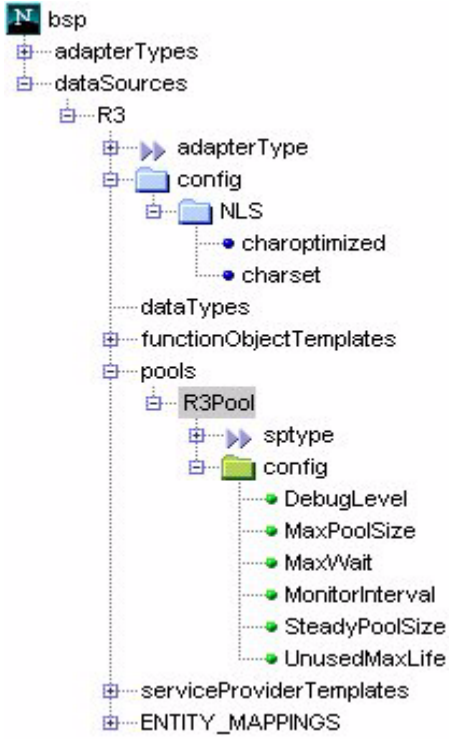
Pool definitions are service provider pools and their controlling attributes. A datasource may define several connection pools, though usually, there is only a single pool. Each service provider defines which pool it uses. Each pool is controlled by a pool configuration block, which contains the following:

- DebugLevel
- MaxPoolSize
- MaxWait
- MonitorInterval
- SteadyPoolSize

- UnusedMaxLife

For more information about pooling, see Chapter 4, “Pooling Concepts”. Figure 3-5 shows the relationship between a service provider and the pool configuration information.

Figure 3-5 Pool Configuration Information



User Mapping Information

User mapping information are definitions of user mapping tables from a web domain to an EIS domain for specific EISs. The user mapping table content is managed via the management console. Refer to your specific iPlanet Enterprise Connector *Administrator’s Guide* for details.

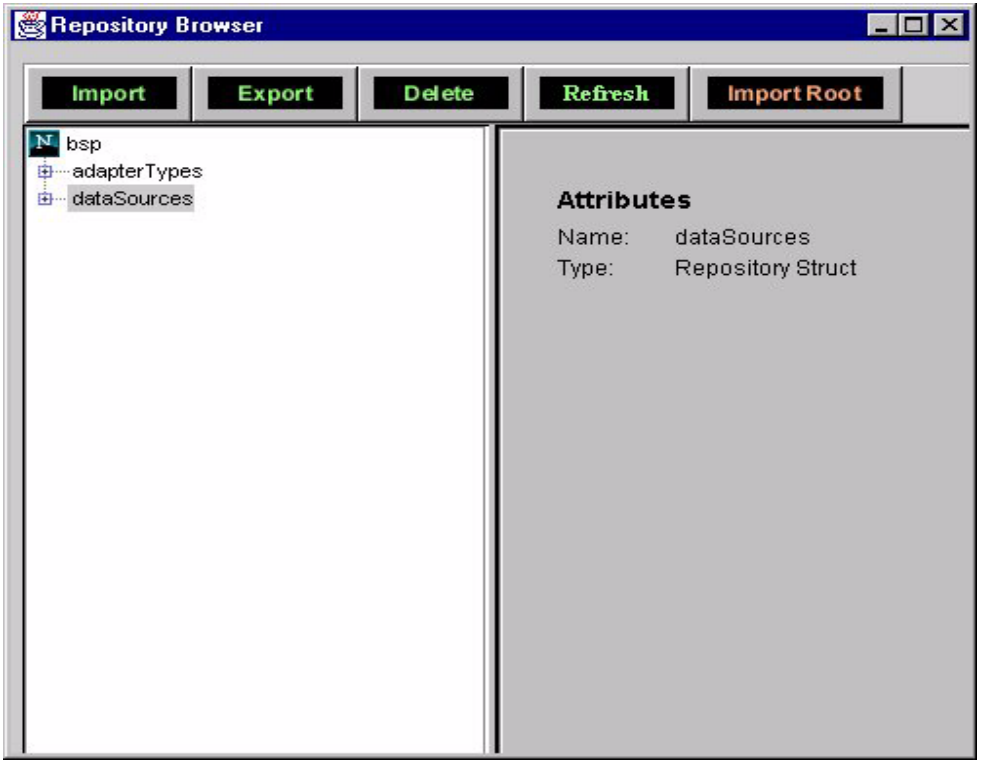
About the UIF Repository Browser

The UIF Repository Browser's main function is to provide a UIF abstraction view of the UIF Repository. In addition, UIF Repository browser provides Import, Export, Delete, Refresh and Import Root functionality on the UIF Repository.

The repository browser is divided into left and right panes. The view hierarchy appears in the left pane and the contents in the right pane. The actual contents depend on which node is selected in the view hierarchy. For the semantic details on the repository content, please refer to your iPlanet Enterprise Connector *Administrator's Guide*.

Figure 3-6 shows an example of the UIF repository browser.

Figure 3-6 Repository Browser Example



View Hierarchy

Click on the folders in the outline view in the left pane to expand and collapse the repository view. Initially, the hierarchy shows the following information:

- Root node (bsp)
- Connector types (adapterTypes)
- Datasources

Icons in the view hierarchy have different meanings, as follows:

Icon	Description
Double arrow	Reference to another object
Bullet	Leaf node, which cannot be traversed further
Folder	Non leaf node

An X appearing on a node indicates the node either does not exist or is invalid. To fix a node, refer to your specific connector documentation for details.

Each node type is represented by a different color. Primitive non data object image (leaf) nodes are yellow. Colors for the data object nodes are identified in the following section.

Data Object views

The repository browser allows the viewing of data object templates, data object types, and data object image folders in different ways. The node color specifies the view currently being displayed. Nodes are color coded, as follows:

Color	Contains
Red	data object type definition
Blue	data object template
Green	data object image

To switch between views, click the right mouse button on a folder and it changes to the desired view. The views available depend on the kinds of data objects available, as follows:

- A data object template can display type information, data, or an image. The data is the type's default value. The image is a value that overrides the default.
- A data object type can display type information or data. The data is the type's default value.
- A data object image can only display its image, which is its value.

Typically, you view the type definition return values and the data object template for values passed as parameters from a Servlet, JSP, or EJB. Refer to Chapter 5, "Programming Model" for more information on data object images, templates, etc.

Using the UIF Repository Browser

The UIF Repository Browser is used to perform the following operations:

- Import a node and its subnodes
- Export a node and its subnodes
- Delete a node and its subnodes
- Refresh the UIF repository browser, so that it reflects changes made to the LDAP storage from the management console
- Import the root node

Import a Node and its Subnodes

A node is imported with its subnodes into the specified repository node. A typical use of this functionality is to import the data source XML prepared on a development machine UIF Repository to a deployment machine UIF Repository.

NOTE An imported XML file describes a repository node or data object node. An XML file cannot be imported from within a data object node. Please refer to your specific Enterprise Connector *Administrator's Guide* for the `repository.dtd` to be used.

To import a node:

1. Select a node from the view hierarchy. When Import process is done, all data from the input XML file is added to the node.
2. Click the Import icon in the menu bar.
3. Specify or browse the XML file to be imported.
4. Specify a log file or keep the default.
5. Click OK to start the import.
6. View the log file for messages or errors in the XML file.

Import Notes:

- The XML file imported must exist in the same directory as the `repository.dtd` file. Otherwise, an error occurs.
- Importing deletes an existing subtree with the same root node as the imported XML file. For example, if `bsp.A.B` already exists and an XML file with root `B` is being imported under `bsp.A`, it first deletes the `bsp.A.B` subtree and then imports the XML subtree specified in the import file under `bsp.A`.

Export a Node and its subnodes

A UIF Repository Node and all its subnodes can be exported to an XML file which is often used to export the datasource configurations on the development machine as XML files. The XML files are imported on to the deployment machine, as part of the deployment process.

NOTE A repository node or data object node may be exported as a whole; but a node within a data object node cannot be exported

To export from a node, select from the view hierarchy the node to export.

1. Click the Export icon in the menu bar.
2. Specify a name of the destination XML file.
3. Specify a log file or keep the default.
4. Click OK to start the export operation.

5. View the log file for messages and errors in the XML file.

Delete a Node and its subnodes

Deleting a node also deletes all subnodes underneath it. For example, a node may need to be deleted because it was created in the wrong hierarchy location.

NOTE A repository node or data object node can be deleted; but a node cannot be deleted from within a data object node.

To delete a node:

1. Select from the view hierarchy the node to be deleted.
2. Click the Delete icon in the menu bar.
3. Click Yes when the warning message appears.
4. View the log file for messages and errors.

Refresh the UIF Repository

Refreshing the UIF Repository Browser brings the Repository browser in sync with the LDAP storage. A typical usage of Refresh is after using the Enterprise Connector Management Consoles. Since the Management Console populate the UIF repository directly, the UIF Repository Browser might go out of sync with the UIF Repository.

As mentioned above, the Refresh functionality brings the UIF Repository browser in sync with the UIF Repository. In order for the UIF runtime to make use of the new content in the Repository, the iPlanet Application Server need to be restarted.

Import the Root Node

To import a root node and its subtrees:

1. Click the Import Root icon in the menu bar.
2. Specify or browse for the XML file that defines the subtree.

3. Specify the log file or keep the default.
4. Click OK to start the import.
5. View the log file for any messages or errors in the XML file.

View the Log

Use any ASCII text editor to view the log file. By default, the log file is called `bspbrowser.log` in the `APPS` subdiretory where UIF is installed.

Troubleshooting Tips

The following errors can occur when importing or exporting XML files:

- If a `FileNotFound` exception is received make sure the given file location is a valid location.
- If a parsing error is received, it may be that the XML file is not validating against `repository.dtd`. Examine and correct the XML file.
- If a `ClassNotFound` error is received, the `xml.jar` file may not be in the classpath. Although the installation setup handles setting the classpath, modify the classpath if the file's location is not in the classpath.

Chapter 4

Pooling Concepts

This chapter describes the concepts for setting up a pooling configuration node in the UIF repository.

The chapter contains the following sections:

- About Pooling
- Bind Durations

About Pooling

Pooling allows the ability to share scarce resources. In UIF, pools are used to share connections to a EIS from the iPlanet Application Server. UIF interacts with the enterprise connectors to allocate and reuse the pools to avoid unnecessary creation and destruction of connections to the EIS. Pools and their configuration information is defined in the repository for each datasource. In other words, each datasource defines its own connection pool which can be individually configured.

An object in a UIF pool represents a connection to an EIS. The maximum number of connections in the object pool may be specified in the repository. A pool typically contains the number of connections required for the application to maintain a steady state.

When an iPlanet Application Server application enables a service provider, the enterprise connector attempts to obtain a connection object from the pool. If the pools maximum size has been reached and no object is available after a specified waiting period, the request times out. The timeout period can also be configured in the repository.

If a pool object remains unused for a period of time, the object is destroyed. You can specify how long to wait before unused objects are destroyed.

The use and longevity of pool objects are monitored by UIF periodically. You can specify the interval in which the monitoring occurs. Typically, the monitoring thread executes more frequently than the waiting period before an object is destroyed. However, setting the monitor interval too small can degrade performance.

Pooling Configuration

The pooling configuration for a service provider is specified with the following characteristics in the repository:

Parameter	Description
SteadyPoolSize	The number of unused objects that are kept in the pool until they time out. (See UnusedMaxLife.) For example, if the pooled objects are host connections, set SteadyPoolSize to the steady state number of connections available from the host server.
MaxPoolSize	The maximum number of objects allowed in the pool. For example, if the pooled objects are host connections, set this number to the peak number of connections available to the server. If the number of objects exceeds SteadyPoolSize, objects are destroyed after they are returned to the pool.
MaxWait	The maximum time, in seconds, a request for a pooled object is held in the queue before the request times out and is destroyed.
UnusedMaxLife	The maximum time, in seconds, that a pooled object remains unused in the pool. After this time, the physical object is destroyed.
MonitorInterval	(Optional) The time, in seconds, that a thread is executed to monitor the current status of the pool. Default is 30 seconds. Typically, setting this number too low degrades performance, but it should be set to a number less than UnusedMaxLife.
DebugLevel	(Optional) Determines type of message logging, as described by the following choices: 0: Logging turned off. 1: Logs only callback messages. 2: Logs all messages.

Bind Durations

A pooled connection is bound to a J2EE component for a specified period of time, called the bind duration, which is specified in a service provider type. Typically, the bind duration is only for the length of time it takes to execute a single method requiring the connection. Some operations, such as beginning a transaction, may require a longer bind duration.

Bind durations are specified in the repository. UIF supports three bind durations, as follows:

Bind Duration	Description
method-bound-poolable	The pooled connection is bound for the duration of the method call.
sp-bound-poolable	The pooled connection is bound for the life of the service provider; then the connection is returned to the pool.
not-poolable	The connection is not poolable. The connection is bound for the life of the service provider and is destroyed when the service provider is destroyed.

Bind Duration Escalation

The bind duration may be changed dynamically by the enterprise connector. For example, at the beginning of a transaction, an enterprise connector may escalate the bind duration from method-bound to sp-bound, and reset the bind duration to method-bound when the transaction completes.

Bind Duration Timeouts

An sp-bound connection cannot be used by another service provider until the connection has been released by the first service provider. If the first service provider does not release the connection in a timely manner, a timeout occurs and the connection is forcibly released and returned to the pool. The timeout prevents a *run away* service provider that failed to call `disable()` from keeping the connection indefinitely.

Chapter 5

Programming Model

This chapter describes the UIF application programming model used inside your J2EE components, for example, Servlets, JSPs and EJBs.

This chapter contains the following sections:

- Data Objects
- Using the UIF API
- Developing J2EE I18N Applications with UIF

Data Objects

A function object is the fundamental unit of interaction with the enterprise connector. Data objects are typically used as input and output parameters to the function objects. The application programmer is responsible for preparing the input data objects, executing the function object, and interpreting the output data object, through the UIF API. The UIF data object interface:

- Presents a unified and abstract representation of EIS data types
- Supports most primitive data types
- Provides the ability to construct and interpret complex data

Data objects are built from primitive data types. In general, a primitive data object contains a primitive value, for example an integer or a string. A complex data object is a list, an array, or a structure.

As part of the data mining process, the administrator populates the repository with the data object definitions. The application programmer needs to follow these definitions to construct, and interpret the input and output data objects when interacting with the EIS. Data object construction and interpretation is described in the “Using the UIF API” on page 86.

NOTE The data objects represent data in a hierarchal fashion and circular references are not allowed. UIF only prevents adding a data object as an attribute of itself, the application programmer is responsible for preventing indirect circular references. If a circular reference is made via another object, it is not discovered when the reference time is established. If a circular reference is used at runtime, unpredictable results occur.

Primitive Objects

A primitive data type object contains a single value of one of the following types:

- Integer, float, double
- Fixed length string or variable length string
- Fixed size byte array or variable size byte array

When a primitive data object is assigned to a list, array, or structure, the data object is unwrapped and its value is copied into the list, array, or structure. The data object itself is not used. When a primitive value is obtained by using an untyped get method, such as `getField()`, `getElem()`, `getAttr()`, or `getCurrent()`, the returned value is wrapped in a primitive data object. In this case, the value is copied, modifying the returned primitive data object does not change the source object.

Strings correspond to the Java string data type. A fixed length string has a maximum length, whereas a variable length string has no length restriction. A fixed size byte array has a maximum size, whereas a variable size byte array has no size restriction.

In general, when a new value replaces an old value, the old value is released. Therefore, when a new value is assigned to a variable length string or a variable size byte array, the old value is released. When a new value is assigned to a fixed length string or a fixed size byte array, the new value is copied over the old one. For a fixed length string, the copied string is truncated to fit if necessary. For a fixed size byte array, a shorter array is zero filled on the right, and a longer array is truncated to fit.

The maximum length of a fixed length string and the maximum size of a fixed size byte array are set when they are first set. For example, when setting a list element to contain a fixed length string, the string's maximum size is set when added to the value. In the following example, the maximum length of the fixed length string set to an element in a list is four characters:

```
list.addElemFString("abcd");
```

Attempting to replace the string with a five character string (abcde), the last character (e) is truncated.

NOTE The length of the fixed length string is its maximum length; not its current length, because its current length may be less than its maximum length. The size of a byte array is its maximum size; not its current size, because the byte array may not be filled to capacity. When The iPlanet Application Server is running in I18N mode, UIF takes care of preparing the Fixed Strings to accommodate the incoming multibyte character set values, based on the datasource character set configuration information. For a detailed description on the default character set, available EIS character set and how to specify them, see your specific iPlanet Enterprise Connector *Developer's* and *Administrator's Guides.*

Structure Objects

Structure objects contain other data objects or primitive values as fields. Each object within the structure object is referred to by a string that represents the field name. Field names have a maximum length of 64 characters.

List Objects

A list object contains data objects or primitive values as elements in the list. These elements can be heterogeneous. Each element within a list object is referred to by an integer that specifies its position in the list object.

Array Objects

An array object, like a list object, contains data objects or primitive values as elements in the object. Array objects inherit from list objects. The difference between an array object and a list object is that the array's elements must be homogeneous. Each element within the array object is referred to by an integer that specifies its position in the array object.

Type Info Objects

Type info objects are structure objects that contain the type information of a data object. For example, a type info object might define the fields in a structure, and their corresponding data types. Instances of data objects can be created from type info objects. These instances each contain a reference to a type info object. Many data objects can share the same type info object.

DataObjectInfo **Type**	**Target Object**
IBSPDataObjectPrimitiveInfo describes the type number, size of value (if type is string or binary), and default value	IBSPDataObjectPrimitive
IBSPDataObjectStructureInfo describes the type info of all fields of the structure. The type info of each field is in turn described by a type info object	IBSPDataObjectStructure
IBSPDataObjectListInfo describes the initial capacity and maximum element count of the list	IBSPDataObjectList
IBSPDataObjectArrayInfo describes the initial capacity, maximum element count and the type info of elements of the array	IBSPDataObjectArray

API Naming Conventions

Most methods in the API conform to a naming convention that specifies

- Kind of operation: typically, get or set
- Target: a list, structure, and so on
- Type of target: an int, string, and so on

For example, the method that sets a list element from a string is `setElemString()`. The following table shows the possible combinations:

Operation	Target	Type
get	NONE (Primitive)	Int
set	Attr (complex dataObject, such as a list, array, or structure); uses path to address attribute	Float
	Elem (List / Array); uses index to address element	Double
	Field (Structure); uses name to address field	String
	Current (Itr); addresses object iterator is currently on.	FString
		Binary
		VBinary
		dataObject
		NONE

Attributes and Paths

In the UIF API, methods of the IBSPDataObject interface do not distinguish between an element in a list or array, or a field in a structure. In this case, the element or field is referred to as an attribute.

The path to an element is its element number, beginning with 0. The path to a field is its field name. Combining element numbers and field names are used to create paths to attributes in complex data objects. For example, a structure field containing a list of elements. In these cases, specifying the path as the individual attributes separated by dots (.). For example, use "`field1.[0]`" to identify the first list element at `field1` in the structure.

Changing Data Types

If an attribute's type is primitive, it cannot change. For example, the following code causes an error because it tries to change the primitive type from an integer to a float:

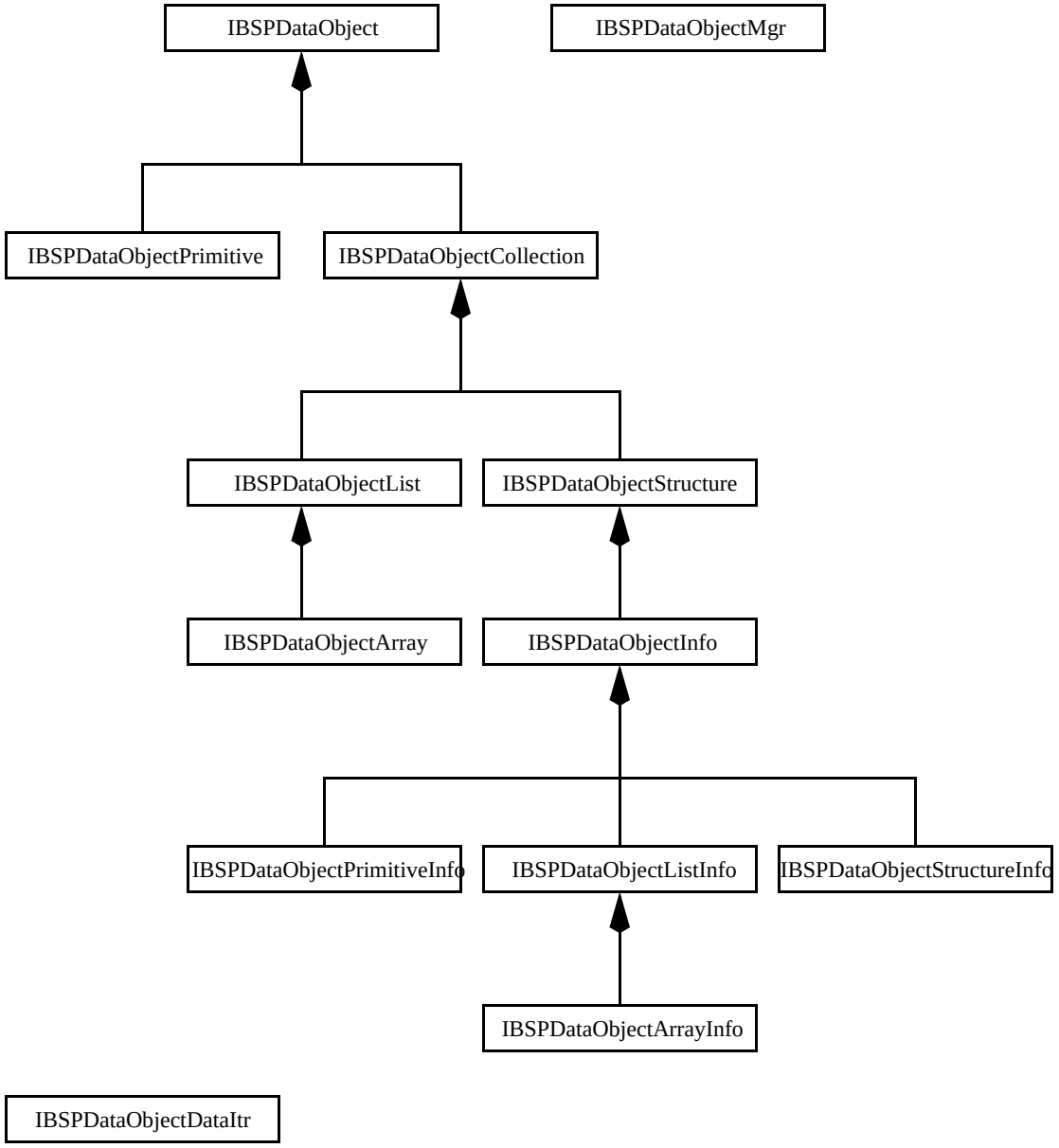
```
list.addElemInt(100); // assume 100 is added to element 1
list.setElemFloat(1, 3.14); // fails because element 1 type is int
```

The following example shows how to change the data type of a non-primitive data:

```
list.addElemDataObject(aStruct); // add a structure to element 1
list.setElemDataObject(1, array); // change to array
```

Interface Hierarchy

The following diagram shows the interface hierarchy for data object interfaces:


IBSPDataObject
IBSPDataObjectMgr
IBSPDataObjectPrimitive
IBSPDataObjectCollection
IBSPDataObjectList
IBSPDataObjectStructure
IBSPDataObjectArray
IBSPDataObjectInfo
IBSPDataObjectPrimitiveInfo
IBSPDataObjectListInfo
IBSPDataObjectStructureInfo
IBSPDataObjectArrayInfo
IBSPDataObjectDataItr

Using the UIF API

The UIF programming model gives fine grained control to the application programs. The usage of the UIF API should broadly follow these steps:

- Acquiring the UIF Runtime Object
- Creating a Service Provider Object
- Creating a Function Object
- Setting Up and Executing the Function Object

The following sections provide examples of how to perform these tasks.

Acquiring the UIF Runtime Object

The runtime object is the entry point into UIF. It is both the object factory and the access point for creating other objects.

The following sample shows how to acquire a runtime object from a Servlet:

```
private IBSPRuntime getRuntime()

{

com.kivasoft.IContext _ctx =
((com.netscape.server.servlet.platformhttp.PlatformServletContext)
getServletContext()).getContext();


   IBSPRuntime ibspruntime = access_cBSPRuntime.getcBSPRuntime
          (_ctx, null, null);
   return ibspruntime;

}
```

Creating a Service Provider Object

The service provider is a logical representation of a connection to an EIS. Typically, the service provider is not bound to a physical connection until it is absolutely necessary. A service provider must be enabled before it can be used.

The following sample shows how to create a service provider object:

```
private IBSPServiceProvider getServiceProvider(IBSPRuntime runtime)
```

```
{
   if (runtime != null)
      return runtime.createServiceProvider("<connector>",
"<connector>_spname");
   else
   return null;
}
```

where, "<connector>" is the datasource name, and "<connector>_spname" is a service provider name defined for that datasource.

Creating a Function Object

A function object is a group of related operations that share a common state. In UIF, a function object needs to be set up and associated with a service provider before the function object can be executed.

The following sample shows how to create a function object:

```
IBSPFunctionObject fn = null;
...
{
   fn = runtime.createFunctionObject("<connector>", "phonebook");
}
```

where, "<connector>" is the datasource name and "phonebook" is the function object name in the repository.

Setting Up and Executing the Function Object

To execute a function object, your Servlet must perform the following actions:

1. Create and enable the service provider
2. Create the function object and associate it to the service provider
3. Prepare the function object and set the input parameters in the datablock
4. Execute the function object
5. Retrieve the output parameters from the function block
6. Disable the service provider

The following sample shows how to set up and execute a function object.

```
IBSPDataObject getCustomerDetail(String custId)
{
IContext ctx = null;
IBSPServiceProvider sp = null;
IBSPRuntime runtime = null;
IBSPDataObject result = null;
try {
   // Get context
   ctx =
((com.netscape.server.servlet.platformhttp.PlatformServletContext)g
etServletContext()).getContext();

   // Get UIF runtime
   runtime = access_cBSPRuntime.getcBSPRuntime(ctx, null, null);

   // Create ServiceProvider 'conn' under datasource 'HR'
   sp = runtime.createServiceProvider("HR", "conn");

   // Enable the ServiceProvider
   IBSPDataobject config = sp.getConfig();
   config.setAttrString("WebUserId", m_UserId);
   sp.enable();

   // Create the target FunctionObject 'getCustomerDetail'
   IBSPFunctionObject  fn = runtime.createFunctionObject("HR",
   "getCustomerDetail");

   // Associate the FunctionObject with the ServiceProvider
   fn.useServiceProvider(sp);

   // Prepare the FunctionObject for execution of operation 'doit'
```

```
    fn.prepare("doit");

    // Set inputs in datablock
    IBSPDataObject data = fn.getDataBlock();
    data.setAttrString("INPUT.custId", custId);

    // Execute the FunctionObject
    fn.execute();

    // Retreive outputs from datablock
    result = data.getAttr("OUTPUT.custDetail");
}

// Catch any exception which may be thrown
catch (BspException)
{
    // BspException actions
}
// Disable the ServiceProvider (regardless of whether an exception
was thrown)
finally
{
     if (sp != null)
    sp.disable();
}
return result;
}
```

Executing Multiple Function Objects

An application may need to execute several functions objects or repeatedly execute a single function object within a Servlet, JSP, or EJB. To repeatedly execute a function object, follow these steps:

1. Prepare the function object
2. Set the input parameters
3. Execute the function object
4. Retrieve the results
5. Loop as many times as needed

To execute several function objects, the application can create and associate multiple function objects with the same service provider at any point, and execute them. Some enterprise connectors support transactions that are started and terminated by an enterprise connector specific function object operation(s). For these enterprise connectors, a transaction context is associated with the service provider instance. *All* function objects using this service provider instance execute as part of the transaction associated with the service provider at that time.

Developing J2EE I18N Applications with UIF

UIF supports the development of J2EE I18N applications using various character sets supported by iPlanet Application Server. To use UIF in I18N mode, please make sure that the iPlanet Application Server is running in International mode. For details on how to run the iPlanet Application Server in International mode, please refer to iPlanet Application Server documentation.

UIF datasources are configured to operate using a specific character set, that is specified in the UIF Repository. UIF uses the datasource character set to prepare the buffer sizes to accommodate multibyte character data with FString data type. For a detailed description on the Enterprise Connectors default character set, allowed operating character set and how to specify them, see your specific iPlanet Enterprise Connector *Developer's* and *Administrator's Guides.*

For details on how to write J2EE I18N applications, please refer to the following documentation:

- Java Servlet API Specification, Version 2.2, located on the web at *http://java.sun.com/products/servlet/2.2/*, sections 5.8 Internationalization (The Request) and 6.4 Internationalization (The Response).

- JavaServer Pages Specification, Version 1.1, located on the web at *http://java.sun.com/products/jsp/download.html*, section 2.7.4 Delivering Localized Contents.
- Designing Enterprise Applications, Version 1.0, located on the web at *http://java.sun.com/j2ee/download.html#blueprints*, section 4.5 Internationalization and Localization.

Chapter 6

Exceptions

This chapter describes the generic exception framework implemented in UIF. It also describes various exception types, exception scenarios, and how to handle them. In addition, refer to your specific iPlanet Enterprise Connector *Developer's Guide* for more details on enterprise connector specific exceptions.

The chapter contains the following sections:

- About UIF Exceptions
- Exception Scenarios

Refer to Chapter 5, "Programming Model" for more information on UIF programming model and suggestions pertaining to the UIF exception handling mechanisms.

About UIF Exceptions

UIF exceptions are used to notify the user of an error. UIF exceptions follow Java exception patterns and have all the characteristics of Java exceptions. UIF throws various types of netscape.bsp.BspException which can be interpreted and extended to provide more meaningful information at the application level.

All UIF exceptions are logged into the iPlanet Application Server logs. The logging of UIF exceptions are enabled by setting log_user_exceptions to 'on' in the iPlanet Application Server registry using the iPlanet Application Server tools. Refer to the iPlanet Application Server documentation for the specifics of changing log levels. Refer to "Related Information" on page 18 for details.

In addition to the generic netscape.bsp.BspException, UIF throws the following data object specific exceptions.

- netscape.bsp.dataobject.exceptions.DataObjectException

- netscape.bsp.dataobject.exceptions.FieldNameIsNullException
- netscape.bsp.dataobject.exceptions.FieldNameTooLongException
- netscape.bsp.dataobject.exceptions.FieldNotFoundException
- netscape.bsp.dataobject.exceptions.IllegalPathException
- netscape.bsp.dataobject.exceptions.IllegalRecursionException
- netscape.bsp.dataobject.exceptions.IndexOutOfBoundsException
- netscape.bsp.dataobject.exceptions.MethodUnsupportedException
- netscape.bsp.dataobject.exceptions.TypeMismatchException

Refer to Chapter 7, "API Reference" for more information on these Exceptions.

UIF exceptions have a wrapped IBSPDataObjectStructure inside which provides the error information. This object is obtained by calling `getInfo()` on the UIF exception. This is an opaque structure that is populated by the exception thrower to provide details on the error state. This structure is not used by most of the exceptions originating from UIF. The enterprise connectors use this feature more concretely to forward the EIS error details. Refer to your specific iPlanet Enterprise Connector *Developer's Guide* for specific exceptions and the error data object structure contents.

Exception Scenarios

UIF exceptions can occur in the following scenarios:

- Application Logic Errors
- UIF Runtime Errors
- Enterprise Connector Errors

Application Logic Errors

Exceptions that usually occur due to application logic errors. The various exceptions with correction information are as follows:

1. DataObject missing

   Reason: The specified path may be wrong or the data object may not be available in the repository.

2. IllegalRecursion

   Reason: Illegal recursion when getting or setting values on a data object.

3. IndexOutOfBounds

   Reason: The array or list index is exceeding the number of elements.

4. IllegalPath

   Reason: Data object, function object, and / or service path is not set correctly.

5. FieldNotFound

   Reason: The data object field is missing in the repository.

6. FieldNameIsNull

   Reason: The field name is null in the repository.

7. FieldNameTooLong

   Reason: The field name specified is too long.

8. MethodUnsupported

   Reason: The method being used is not supported.

9. TypeMismatch

   Reason: The value being set does not match the type specified in the repository.

10. Invalid datasource / domain

    Reason: The datasource or domain is not specified in the repository.

11. Repository node not found

    Reason: The repository node being accessed is not found in the repository.

12. Unable to create map dataobject

    Reason: Unable to create the map object for a specific data object.

13. Map is not array

    Reason: The map being specified is not an array.

14. Bad Mapelement type

    Reason: The element type specified in the data object and data object map do not match.

15. Invalid MapStruct

    Reason: The data object map specified for the data object is invalid.

16. Access not allowed for this user

    Reason: The web user and the datasource user match is not successful. The access is not allowed for the specified user.

17. Function Object not prepared yet

    Reason: The web user is trying to use a function object that has not been prepared yet.

18. Object instance is not valid

    Reason: The object instance being accessed is invalid or is no longer in memory.

19. Invalid repository content

    Reason: The respository content being accessed is invalid.

20. Invalid repository content for datasource

    Reason: The repository content being access is invalid for the datasource being used.

UIF Runtime Errors

Exceptions that usually occur due to UIF runtime errors. The various exceptions with correction information are as follows:

1. Metadata service missing

   Reason: The UIF metadata service is not loaded properly.

2. LDAPRepository missing

   Reason: The LDAP repository is not available to read contents from, it may be down or the application server cannot reach it.

3. Cannot create repository with LDAP parameters: `hostname`, `port`, `userdn`, `password`, and `bspbasedn`

   Reason: UIF cannot access the repository with the configuration parameters such as host name, port, username, password and bsp base node.

4. Cannot create pool

   Reason: UIF cannot create the pool based on the pool path. The configuration may be wrong or the pool size is set to 0.

5. Pooled connection cannot be reserved (likely, the maximum number of concurrent connections is reached)

   Reason: UIF is unable to reserve a pooled connection. The system may have reached the maximum number of concurrent users.

6. Cannot find pool

   Reason: Cannot find the specified pool.

7. Service provider has not been enabled or has timed out

   Reason: The service provider being accessed has not been enabled or has timed out.

8. Cannot retrieve LDAP configuration data

   Reason: UIF is unable to retrieve LDAP configuration data.

9. Cannot find datasource instance with name

   Reason: UIF cannot find the datasource instance with the specified name.

10. Service provider has timed out with state

    Reason: The service provider with certain state content has timed out and is no longer available.

11. Repository instance missing

    Reason: The repository instance is not accessible by UIF and may not be loaded properly.

12. Repository manager unavailable

    Reason: The repository manager instance is not accessible by UIF and may not be loaded properly.

Enterprise Connector Errors

Exceptions that usually occur due to enterprise connector errors. Please refer to your enterprise connector documentation for more specific enterprise connector exceptions. The various exceptions with correction information are as follows:

1. Cannot create the datasource instance with name

   Reason: UIF is unable to create the datasource instance being specified. The enterprise connector may not be available or not loaded properly.

2. Cannot find extension

   Reason: UIF cannot find a specific enterprise connector extension. The connector may not be available or not loaded properly.

UIF Exceptions Handling

Most exceptions that originate from the UIF runtime are due to various repository and access errors. These are data path missing errors, invalid content errors, illegal access errors, etc. These development time errors need to be resolved by taking the necessary corrective actions in the repository or the user code. Use the *reason* presented to make corrective actions.

Runtime specific errors need to have programmatic correction steps to exit from the error state gracefully. A few of the exception thrown by UIF are due to either missing a specific service or LDAP related errors explained above. These error may not be taken care of programmatically.

The crucial exceptions to handle programmatically are thrown by the enterprise connector. Enterprise connector specific exceptions extend `netscape.bsp.BspException`. Use `getInfo()` for the enterprise connector specific exceptions to interpret the results and take corrective actions accordingly. Refer to your specific iPlanet Enterprise Connector *Developer's Guide* for detailed error description contents.

Refer to Chapter 5, "Programming Model" for more information on UIF programming model and suggestions pertaining to the UIF exception handling mechanisms.

Chapter 7

API Reference

This chapter describes the interfaces and methods used from your Servlet, JSP, or EJB to access and manipulate data stored on an EIS system.

This chapter contains the following sections:

- IBSPRuntime Interface
- IBSPServiceProvider Interface
- IBSPFunctionObject Interface
- IBSPDataObjectMgr Interface
- IBSPDataObject Interface
- IBSPDataObjectPrimitive Interface
- IBSPDataObjectPrimitiveInfo Interface
- IBSPDataObjectCollection Interface
- IBSPDataObjectList Interface
- IBSPDataObjectArray Interface
- IBSPDataObjectStructure Interface
- IBSPDataObjectStructureInfo Interface
- IBSPDataObjectInfo Interface
- IBSPDataObjectListInfo Interface
- IBSPDataObjectArrayInfo Interface
- IBSPDataObjectDataItr Interface
- BspException Class

- DataObjectException Class
- FieldNameIsNullException Class
- FieldNameTooLongException Class
- FieldNotFoundException Class
- IllegalPathException Class
- IllegalRecusionException Class
- IndexOutOfBoundsException Class
- MethodUnsupportedException Class
- TypeMismatchException Class

IBSPRuntime Interface

The `IBSPRuntime` interface defines the methods to create service provider and function objects.

Package

`netscape.bsp.runtime`

Methods

Method	Description
`createFunctionObject()`	Creates a function object.
`createServiceProvider()`	Creates a service provider object.
`getServerContext()`	Retrieves the server's context object.

createFunctionObject()

Creates a function object.

Syntax

```
public
netscape.bsp.runtime.IBSPFunctionObject createFunctionObject(
       java.lang.String pAdapterName,
       java.lang.String pFunctionObjectName);
```

pAdapterName – A string that specifies the connector name.

pFunctionObjectName – A string that specifies the function object name.

Usage

Specify the datasource name and a function object name from the repository.

Example

```
IBSPFunctionObject fn = null;

if( (runtime != null) && (sp != null) )

   {

      fn = runtime.createFunctionObject("<connector>",
"phonebook");

   }
```

Return Value

An `IBSPFunctionObject`.

createServiceProvider()

Creates a service provider object.

Syntax

```
public
netscape.bsp.runtime.IBSPServiceProvider createServiceProvider(
       java.lang.String pAdapterName,
       java.lang.String pServiceProviderName);
```

pAdapterName – A string that specifies the connector name.

pServiceProviderName – A string that specifies the service provider name.

Usage

Specify the datasource name and a service provider name from the repository.

Example

```
IBSPServiceProvider sp = null;

if (runtime != null)
```

```
{
   sp = runtime.createServiceProvider("<connector>",
"<connector>_sp1");
}
```

Return Value
An `IBSPServiceProviderObject` object.

getServerContext()

Retrieves the server's context object.

Syntax

```
public com.kivasoft.IContext getServerContext();
```

Return Value
The `IContext` object associated with this runtime object.

IBSPServiceProvider Interface

The `IBSPServiceProvider` interface defines the methods to enable or disable a connection to an EIS server from a Servlet, JSP, or EJB.

Package

`netscape.bsp.runtime`

Methods

Method	Description
`disable()`	Disables the EIS connection.
`enable()`	Enables the EIS connection.
`extendBindDuration()`	Extends the bind duration.
`getConfig()`	Determines the configuration of the EIS connection.
`isEnabled()`	Determines whether the EIS connection is enabled.

disable()

Disables the EIS connection.

Syntax

```
public int disable();
```

Usage

The `disable()` method causes the connection with the EIS to be dropped. The application needs to disable `sp-bound` connections after they are no longer needed. After calling the `disable()` method, the connection is no longer valid and the connection is returned to the pool. The application can call `enable()` to reestablish the connection.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

enable()

Enables the EIS connection.

Syntax

```
public int enable();
```

Usage

The `enable()` method validates the logical connection with the EIS system and establishes a physical connection. The application must enable the connection before it associates a service provider with a function object. For `method-bound` connections, the connection is automatically released after the method call.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

extendBindDuration()

Extends the bind duration.

Syntax
```
public int extendBindDuration();
```

Usage
The `extendBindDuration()` method increases the bind duration timeout value beyond the bind duration specified in the repository. Call this method when the application expects a long transaction and want to hold the same physical connection. For information about bind durations, see “Bind Durations” on page 73.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

getConfig()
Determines the configuration of the EIS connection.

Syntax
```
public netscape.bsp.dataobject.IBSPDataObject getConfig();
```

Usage
Use this method to set up configuration information for an EIS connection.

Return Value
An `IBSPDataObject` object that contains the configuration information.

isEnabled()
Determines whether the EIS connection is enabled.

Syntax
```
public boolean isEnabled();
```

Return Value
A boolean value that is true if the connection is enabled; otherwise, it is false.

IBSPFunctionObject Interface

The `IBSPFunctionObject` interface defines the methods the application needs to specify a service provider which prepares and executes an EIS request to obtain properties and data from the EIS.

Package

`netscape.bsp.runtime`

Methods

Method	Description
`execute()`	Executes an EIS request.
`getDataBlock()`	Retrieves a data block from the EIS system.
`getProperties()`	Determines the EIS system properties.
`getServiceProvider()`	Determines the service provider.
`prepare()`	Prepares an EIS request.
`useServiceProvider()`	Specifies the service provider to use.

execute()

Executes an EIS request.

Syntax

```
public int execute();
```

Usage

Calls the `prepare()` method before calling `execute()`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

getDataBlock()

Retrieves a data block from the EIS system.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject getDataBlock();
```

Return Value

An `IBSPDataObject` object that contains the data block.

getProperties()

Determines the EIS system properties.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject getProperties();
```

Return Value

An `IBSPDataObject` object that contains the properties for the current operation.

getServiceProvider()

Determines the service provider.

Syntax

```
public netscape.bsp.dataobject.IBSPServiceProvider
getServiceProvider();
```

Return Value

An `IBSPServiceProvider` object that represents a connection to the EIS.

prepare()

Prepares an EIS request.

Syntax

```
public int prepare(java.lang.String pOperation);
```

pOperation – A string that specifies the operation to perform.

Usage

Specify an operation by name from the repository. The `prepare()` method sets up the data block and properties for the specified operation and causes the specified operation to become the current operation for this function object.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectMgr Interface

The `IBSPDataObjectMgr` interface implements a method for examining the contents of a data object. The following example shows how to create an reference to an `IBSPDataObjectMgr` object:

```
netscape.bsp.dataobject.IBSPDataObjectMgr doMgr =
netscape.bsp.dataobject.access_cDataObject.getcDataObject(
       context, null, null);
```

NOTE Currently, data object creation methods are not exposed. The application can only create a data object by creating the object info in the repository and accessing the repository.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`dumpDataObject()`	Displays the data contained in a data object.

dumpDataObject()

Displays the data contained in a data object on the console.

Syntax

```
public int dumpDataObject(
   netscape.bsp.dataobject.IBSPDataObject dataObject);
```

dataObject – The `IBSPDataObject` object whose data the application wants to display.

Usage

Use the `dumpDataObject()` method for debugging. The data is displayed on the console.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObject Interface

The `IBSPDataObject` interface implements methods for accessing data objects and their attributes.

Package

`netscape.bsp.dataobject`

Attribute Access Methods

Method	Description
`getAttr()`	Determines the attribute at the specified path.
`getAttrBinary()`	Determines the value of a byte array attribute.
`getAttrDouble()`	Determines the value of a double precision floating point attribute.
`getAttrFloat()`	Determines the value of a floating point attribute.
`getAttrFString()`	Determines the value of a fixed length string attribute.
`getAttrInt()`	Determines the value of an integer attribute.
`getAttrString()`	Determines the value of a string attribute.
`getAttrVBinary()`	Determines the value of a variable size byte array attribute.
`setAttrBinary()`	Sets the value of a byte array attribute.
`setAttrDataObject()`	Sets the value of an attribute to a data object.
`setAttrDataObjectList()`	Sets the value of an attribute to a new empty list.
`setAttrDataObjectStructure()`	Sets the value of an attribute to a new empty structure.

Method	Description
`setAttrDouble()`	Sets the value of a double precision floating point attribute.
`setAttrFloat()`	Sets the value of a floating point attribute.
`setAttrFString()`	Sets the value of a fixed length string attribute.
`setAttrInt()`	Sets the value of an integer attribute.
`setAttrString()`	Sets the value of a string attribute.
`setAttrVBinary()`	Sets the value of a variable size byte array attribute.

Other Methods

Method	Description
attrExists()	Determines whether an attribute exists in the instance.
`attrIsDefined()`	Determines whether an attribute is defined in the repository.
`copy()`	Creates a copy of this object.
`equal()`	Compares with another data object to determine if their values equal.
`getAttrBinarySize()`	Determines the current size of a byte array attribute.
`getAttrCount()`	Determines the number of attributes in the data object.
`getAttrFStringMaxLen()`	Determines the maximum length of a fixed size string attribute.
`getAttrFStringSize()`	Determines the current size of a fixed size string attribute.
`getAttrStringSize()`	Determines the current size of a string attribute.
`getAttrType()`	Determines the data type of an attribute.
`getAttrVBinarySize()`	Determines the current size of a variable byte array attribute.
`getDataItr()`	Retrieves the data iterator for this object.
`getMetadataItr()`	Retrieves the meta data iterator for this object.
`getTypeInfo()`	Determines the type information for this object.
`isAttrType()`	Determines whether an attribute's type matches the specified type.
`isType()`	Determines whether the type matches the specified type.
`removeAttr()`	Removes an attribute.

attrExists()

Determines whether an attribute exists in the instance.

Syntax

```
public boolean attrExists(java.lang.String path);
```

path – The path to a data attribute the application wants to check.

Usage

Specify the path for the data object attribute, that needs checking.

Return Value

A boolean value that is true if the attribute exits; otherwise, it is false.

attrIsDefined()

Determines whether an attribute is defined in the repository.

Syntax

```
public boolean attrIsDefined(java.lang.String path);
```

path – The path to a data attribute the application wants to check.

Usage

Specify the path for the data object attribute, that needs checking.

Return Value

A boolean value that is true if the attribute is defined; otherwise, it is false.

copy()

Creates a copy of this object.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject copy();
```

Usage

The `copy()` method performs a deep copy.

Return Value

A copy of the data object.

getAttr()

Determines the attribute at the specified path.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject
   getAttr(java.lang.String path);
```

path – The path to a data attribute the application wants to check.

Usage

Specify the path of the Data Object attribute to obtain the attribute value.

Return Value

An `IBSPDataObject` object that contains the attribute.

getAttrBinary()

Determines the value of a byte array attribute.

Syntax

```
public byte[] getAttrBinary(java.lang.String path);
```

path – The path to a data attribute the application wants to check.

Return Value

A byte array that contains the value.

getAttrBinarySize()

Determines the current size of a byte array attribute.

Syntax

```
public int getAttrBinarySize(java.lang.String path);
```

path – The path to a byte array attribute whose size the application wants to obtain.

Return Value

An integer that contains the size, in bytes.

getAttrCount()

Determines the number of attributes in the data object.

Syntax

```
public int getAttrCount();
```

Return Value

An integer that specifies the number of attributes.

getAttrDouble()

Determines the value of a double precision floating point attribute.

Syntax

```
public double getAttrDouble(java.lang.String path);
```

path – The path to a data attribute the application wants to check.

Return Value

A double precision floating point number that contains the value.

getAttrFloat()

Determines the value of a floating point attribute.

Syntax
```
public float getAttrFloat(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to obtain.

Return Value
A floating point number that contains the value.

getAttrFString()
Determines the value of a fixed length string attribute.

Syntax
```
public java.lang.String getAttrFString(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to obtain.

Return Value
A string that contains the value.

getAttrFStringMaxLen()
Determines the maximum length of a fixed size string attribute.

Syntax
```
public int getAttrFStringMaxLen(java.lang.String path);
```

path – The path to a fixed length string attribute whose maximum length the application wants to obtain.

Return Value
An integer that contains the maximum length, in bytes.

getAttrInt()
Determines the value of an integer attribute.

Syntax
```
public int getAttrInt(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to obtain.

Return Value
An integer that contains the value.

getAttrString()

Determines the value of a string attribute.

Syntax

```
public java.lang.String getAttrString(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to obtain.

Return Value

A string that contains the value.

getAttrType()

Determines the data type of an attribute.

Syntax

```
public int getAttrType(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to determine.

Usage

The attribute type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

The application should always use a constant instead of a value in its Servlet, JSP, or EJB.

Return Value

An integer that contains the type number.

getAttrVBinary()

Determines the value of a variable size byte array attribute.

Syntax

```
public byte[] getAttrVBinary(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to obtain.

Return Value

A byte array that contains the value.

getAttrVBinarySize()

Determines the current size of a variable byte array attribute.

Syntax

```
public int getAttrVBinarySize(java.lang.String path);
```

path – The path to a variable length byte array attribute whose size the application wants to obtain.

Return Value

An integer that contains the size, in bytes.

getDataItr()

Retrieves the data iterator for this object.

Syntax

```
public
netscape.bsp.dataobject.IBSPDataObjectDataItr getDataItr();
```

Usage

For information about the `IBSPDataObjectDataItr` interface, see “IBSPDataObjectDataItr Interface” on page 163.

Return Value

An `IBSPDataObjectDataItr` object.

getTypeInfo()

Determines the type information for this object.

Syntax

```
public
netscape.bsp.dataobject.IBSPDataObjectInfo getTypeInfo();
```

Return Value

An `IBSPDataObjectInfo` object.

isAttrType()

Determines whether an attribute's type matches the specified type.

Syntax

```
public
boolean isAttrType(java.lang.String path, int typeNumber);
```

typeNumber – The specified type.

Usage

The attribute type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

Return Value

A boolean value that is true if the types match; otherwise, it is false.

isType()

Determines whether the type matches the specified type.

Syntax

```
public boolean isType(int typeNumber);
```

typeNumber – The specified type.

Usage

The attribute type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

Return Value

A boolean value that is true if the types match; otherwise, it is false.

removeAttr()

Removes an attribute.

Syntax

```
public int removeAttr(java.lang.String path);
```

path – The path to a data attribute the application wants to remove.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrBinary()

Sets the value of a byte array attribute.

Syntax

```
public int setAttrBinary(java.lang.String path, byte[] buf);
```

path – The path to a data attribute whose value the application wants to set.

buf – The value with which to set the attribute.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrDataObject()

Sets the value of an attribute to a data object.

Syntax

```
public int setAttrDataObject(java.lang.String path,
       netscape.bsp.dataobject.IBSPDataObject value);
```

path – The path to a data attribute whose value the application wants to set.

value – The object that the application wants to associate with the attribute.

Usage
A value for a primitive data type is unwrapped and copied when it is set in the object; in other words, the value is stored in the object, not an object that contains the value. Otherwise, the data object itself is set to the value.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrDataObjectList()

Sets the value of an attribute to a new empty list.

Syntax

```
public int setAttrDataObjectList(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to set.

Usage

An empty list is created at the specified path.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrDataObjectStructure()

Sets the value of an attribute to a new empty structure.

Syntax

```
public int setAttrDataObjectStructure(java.lang.String path);
```

path – The path to a data attribute whose value the application wants to set.

Usage

An empty structure is created at the specified path.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrDouble()

Sets the value of a double precision floating point attribute.

Syntax

```
public int setAttrDouble(java.lang.String path, double value);
```

path – The path to a data attribute whose value the application wants to set.

value – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrFloat()

Sets the value of a floating point attribute.

Syntax

```
public int setAttrFloat(java.lang.String path, float value);
```

path – The path to a data attribute whose value the application wants to set.

value – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrFString()

Sets the value of a fixed length string attribute.

Syntax

```
public int setAttrFString( java.lang.String path,
       java.lang.String stringValue);
```

path – The path to a data attribute whose value the application wants to set.

stringValue – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrInt()

Sets the value of an integer attribute.

Syntax

```
public int setAttrInt(java.lang.String path, int intValue);
```

path – The path to a data attribute whose value the application wants to set.

intValue – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrString()

Sets the value of a string attribute.

Syntax

```
public int setAttrString(java.lang.String path,
       java.lang.String stringValue);
```

path – The path to a data attribute whose value the application wants to set.

stringValue – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setAttrVBinary()

Sets the value of a variable size byte array attribute.

Syntax

```
public int setAttrVBinary(java.lang.String path, byte[] buf);
```

path – The path to a data attribute whose value the application wants to set.

buf – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectPrimitive Interface

The `IBSPDataObjectPrimitive` interface defines methods for accessing primitive data objects, which include integers, floating point numbers, double precision floating point numbers, byte arrays, and strings.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`getBinary()`	Determines the value of a byte array.
`getBinarySize()`	Determines the size of a byte array.
`getDouble()`	Determines the value of a double precision floating point number.
`getFloat()`	Determines the value of a floating point number.
`getFString()`	Determines the value of a fixed length string.
`getFStringMaxLen()`	Determines the maximum length of a fixed-size string.
`getInt()`	Determines the value of an integer.
`getString()`	Determines the value of a string.
`getVBinary()`	Determines the value of a variable size binary number.
`getVBinarySize()`	Determines the size of a variable byte array.
`setBinary()`	Sets the value of a binary number.
`setDouble()`	Sets the value of a double precision floating point number.
`setFloat()`	Sets the value of a floating point number.

Method	Description
`setFString()`	Sets the value of a fixed length string.
`setInt()`	Sets the value of an integer.
`setString()`	Sets the value of a string.
`setVBinary()`	Sets the value of a variable size byte array.

getBinary()

Determines the value of a byte array.

Syntax

```
public byte[] getBinary();
```

Return Value

A byte array that contains the value.

getBinarySize()

Determine the size of a byte array.

Syntax

```
public int getBinarySize();
```

Return Value

An integer that contains the size, in bytes.

getDouble()

Determines the value of a double precision floating point number.

Syntax

```
public double getDouble();
```

Return Value

A double precision floating point number that contains the value.

getFloat()

Determines the value of a floating point number.

Syntax

```
public float getFloat();
```

Return Value
A floating point number that contains the value.

getFString()

Determines the value of a fixed length string.

Syntax
```
public java.lang.String getFString();
```

Return Value
A string that contains the value.

getFStringMaxLen()

Determines the maximum length of a fixed size string.

Syntax
```
public int getFStringMaxLen();
```

Return Value
An integer that contains the maximum length, in bytes.

getInt()

Determines the value of an integer.

Syntax
```
public int getInt();
```

Return Value
An integer that contains the value.

getString()

Determines the value of a string.

Syntax
```
public java.lang.String getString();
```

Return Value
A string that contains the value.

getVBinary()

Determines the value of a variable size binary number.

Syntax

```
public byte[] getVBinary();
```

Return Value

A byte array that contains the value.

getVBinarySize()

Determine the size of a variable byte array.

Syntax

```
public int getVBinarySize();
```

Return Value

An integer that contains the size, in bytes.

setBinary()

Sets the value of a binary number.

Syntax

```
public int setBinary(byte[] buf);
```

buf – The value with which to set the object.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setDouble()

Sets the value of a double precision floating point number.

Syntax

```
public int setDouble(double value);
```

value – The value with which to set the object.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFloat()

Sets the value of a floating point number.

Syntax
```
public int setFloat(float value);
```

value – The value with which to set the object.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFString()

Sets the value of a fixed length string.

Syntax
```
public int setFString(java.lang.String stringValue);
```

stringValue – The value with which to set the object.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setInt()

Sets the value of an integer.

Syntax
```
public int setInt(int intValue);
```

intValue – The value with which to set the object.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setString()

Sets the value of a string.

Syntax

```
public int setString(java.lang.String stringValue);
```

stringValue – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setVBinary()

Sets the value of a variable size byte array.

Syntax

```
public int setVBinary(byte[] buf);
```

buf – The value with which to set the object.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectPrimitiveInfo Interface

The `IBSPDataObjectPrimitiveInfo` interface defines methods for getting and setting the default values and lengths of primitive data types.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`getPrimDefault()`	Determines the default primitive data object.
`getPrimDefaultBinary()`	Determines the default value of a byte array.
`getPrimDefaultDataObject()`	Determines the default primitive data object.
`getPrimDefaultDouble()`	Determines the default value of a double precision floating point number.
`getPrimDefaultFloat()`	Determines the default value of a floating point number.
`getPrimDefaultFString()`	Determines the default value of a fixed length string.
`getPrimDefaultFStringMaxLen()`	Determines the default maximum length of a fixed-size string.
`getPrimDefaultInt()`	Determines the default value of an integer.
`getPrimDefaultVBinary()`	Determines the default value of a variable size byte array.
`getPrimSize()`	Determines the size of the defined primitive data object.
`getPrimType()`	Determines the data type of the defined primitive data object.
`setPrimDefaultBinary()`	Sets the default value of a binary number.
`setPrimDefaultDataObject()`	Sets the default primitive data object.
`setPrimDefaultDouble()`	Sets the default value of a double precision floating point number.
`setPrimDefaultFloat()`	Sets the default value of a floating point number.
`setPrimDefaultFString()`	Sets the default value of a fixed length string.
`setPrimDefaultInt()`	Sets the default value of an integer.
`setPrimDefaultString()`	Sets the default value of a string.
`setPrimDefaultVBinary()`	Sets the default value of a variable size byte array.

getPrimDefault()

Determines the default primitive data object.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObjectPrimitive
       getPrimDefault();
```

Return Value
The default `IBSPDataObjectPrimitive` object.

getPrimDefaultBinary()
Determines the default value of a byte array.

Syntax
```
public byte[] getPrimDefaultBinary();
```

Return Value
A byte array that contains the value.

getPrimDefaultDataObject()
Determines the default primitive data object.

Syntax
```
public netscape.bsp.dataobject.IBSPDataObjectPrimitive
       getPrimDefaultDataObject();
```

Return Value
The default `IBSPDataObjectPrimitive` object.

getPrimDefaultDouble()
Determines the default value of a double precision floating point number.

Syntax
```
public double getPrimDefaultDouble();
```

Return Value
A double precision floating point number that contains the value.

getPrimDefaultFloat()
Determines the default value of a floating point number.

Syntax
```
public float getPrimDefaultFloat();
```

Return Value
A floating point number that contains the value.

getPrimDefaultFString()

Determines the default value of a fixed length string.

Syntax

```
public java.lang.String getPrimDefaultFString();
```

Return Value

A string that contains the value.

getPrimDefaultFStringMaxLen()

Determines the default maximum length of a fixed size string.

Syntax

```
public int getPrimDefaultFStringMaxLen();
```

Return Value

An integer that contains the size, in bytes.

getPrimDefaultInt()

Determines the default value of an integer.

Syntax

```
public int getPrimDefaultInt();
```

Return Value

An integer that contains the value.

getPrimDefaultVBinary()

Determines the default value of a variable size byte array.

Syntax

```
public byte[] getPrimDefaultVBinary();
```

Return Value

A byte array that contains the value.

getPrimSize()

Determines the size of the defined primitive data object.

Syntax
```
public int getPrimSize();
```

Return Value
An integer that contains the size, in bytes.

getPrimType()
Determines the data type of the defined primitive data object.

```
public int getPrimType();
```

Usage
The primitive type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

The application should always use a constant instead of a value in its Servlet, JSP, or EJB.

Return Value
An integer that contains the type number.

setPrimDefaultBinary()
Sets the default value of a binary number.

Syntax
```
public int setPrimDefaultBinary(byte[] buf);
```

buf – The default value with which to set the default.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultDataObject()

Sets the default primitive data object.

Syntax

```
public int setPrimDefaultDataObject(
       netscape.bsp.dataobject.IBSPDataObjectPrimitive value);
```

value – The value with which to set the default.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultDouble()

Sets the default value of a double precision floating point number.

Syntax

```
public int setPrimDefaultDouble(double value);
```

value – The value with which to set the default.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultFloat()

Sets the default value of a floating point number.

Syntax

```
public int setPrimDefaultFloat(float value);
```

value – The value with which to set the default.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultFString()

Sets the default value of a fixed length string.

Syntax
```
public int setPrimDefaultFString(java.lang.String value);
```

stringValue – The value with which to set the default.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultInt()

Sets the default value of an integer.

Syntax
```
public int setInt(int intValue);
```

intValue – The value with which to set the default.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultString()

Sets the default value of a string.

Syntax
```
public int setString(java.lang.String stringValue);
```

stringValue – The value with which to set the default.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setPrimDefaultVBinary()

Sets the default value of a variable size byte array.

Syntax

```
public int setVBinary(byte[] buf);
```

buf – The value with which to set the default.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectCollection Interface

The `IBSPDataObjectCollection` interface defines an interface from which collections, such as data object lists and structures are derived. The interface currently has no methods.

IBSPDataObjectList Interface

The `IBSPDataObjectList` interface defines methods for accessing elements of list data objects. The interface defines an ordered list in which new elements are added to the end.

NOTE An element must be added to the list before the element can be set.

Package

`netscape.bsp.dataobject`

Access Methods

Method	Description
`getElem()`	Determines the value of an element.
`getElemBinary()`	Determines the value of a byte array element.
`getElemDouble()`	Determines the value of a double precision floating point number element.
`getElemFloat()`	Determines the value of a floating point number element.
`getElemFString()`	Determines the value of a fixed length string element.
`getElemDataObject()`	Determines the data object stored in an element.
`getElemInt()`	Determines the value of an integer element.
`getElemString()`	Determines the value of a string element.
`getElemVBinary()`	Determines the value of a variable size byte array element.
`setElemBinary()`	Sets the value of a binary number element.
`setElemDataObject()`	Sets the data object stored in an element.
`setElemDataObjectList()`	Sets an element to a new empty list.
`setElemDataObjectStructure()`	Sets an element to a new empty structure.
`setElemDouble()`	Sets the value of a double precision floating point number element.
`setElemFloat()`	Sets the value of a floating point number element.
`setElemFString()`	Sets the value of a fixed length string element.
`setElemInt()`	Sets the value of an integer element.
`setElemString()`	Sets the value of a string element.
`setElemVBinary()`	Sets the value of a variable size byte array element.

Element Insertion and Deletion Methods

Method	Description
`addElemBinary()`	Adds a byte array as an element in the list.
`addElemDataObject()`	Adds a data object as an element in the list.
`addElemDataObjectList()`	Adds a new empty list as an element in the list.
`addElemDataObjectStructure()`	Adds a new empty structure as an element in the list.
`addElemDouble()`	Adds a double precision floating point number as an element in the list.
`addElemFloat()`	Adds a floating point number as an element in the list.

Method	Description
`addElemFString()`	Adds a fixed length string as an element in the list.
`addElemInt()`	Adds an integer number as an element in the list.
`addElemString()`	Adds a string as an element in the list.
`addElemVBinary()`	Adds a variable length byte array as an element in the list.
`removeElem()`	Removes an element in the list.

Other Methods

Method	Description
`elemExists()`	Determines whether an element exists in the instance.
`elemIsDefined()`	Determines whether an element is defined in the repository.
`getElemBinarySize()`	Determines the size of a byte array element.
`getElemCount()`	Determines the current number of elements in the list.
`getElemFStringMaxLen()`	Determines the maximum length of a fixed length string element.
`getElemType()`	Determines the type of an element.
`getElemVBinarySize()`	Determines the size of a variable byte array element.
`getMaxElemCount()`	Determines the maximum number of elements in the list.
`isElemType()`	Compares the element with the specified type.
`setMaxElemCount()`	Specifies the maximum number of elements in the list.

addElemBinary()

Adds a byte array as an element in the list.

Syntax

```
public int addElemBinary(byte[] buf);
```

buf – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemDataObject()

Adds a data object as an element in the list.

Syntax

```
public int addElemDataObject
       (netscape.bsp.dataobject.IBSPDataObject value);
```

value – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemDataObjectList()

Adds a new empty list as an element in the list.

Syntax

```
public int addElemDataObjectList();
```

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemDataObjectStructure()

Adds a new empty structure as an element in the list.

Syntax

```
public int addElemDataObjectStructure();
```

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemDouble()

Adds a double precision floating point number as an element in the list.

Syntax

```
public int addElemDouble(double value);
```

value – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemFloat()

Adds a floating point number as an element in the list.

Syntax

```
public int addElemFloat(float value);
```

value – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemFString()

Adds a fixed length string as an element in the list.

Syntax

```
public int addElemFString(java.lang.String value);
```

value – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemInt()

Adds an integer number as an element in the list.

Syntax

```
public int addElemInt(int value);
```

value – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemString()

Adds a string as an element in the list.

Syntax

```
public int addElemString(java.lang.String value);
```

value – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

addElemVBinary()

Adds a variable length byte array as an element in the list.

Syntax

```
public int addElemVBinary(byte[] buf);
```

buf – The value with which to set the element.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

elemExists()

Determines whether an element exists in the instance.

Syntax

```
public boolean elemExists(int index);
```

index – The position of the element to test.

Usage

An element exists if its value exists in the object. The first index position in the list is `0`.

Return Value

A boolean value that is true if the element exists; otherwise, it is false.

elemIsDefined()

Determines whether an element is defined in the repository.

Syntax

```
public boolean elemIsDefined(int index);
```

index – The position of the element to test.

Usage

An element is defined if its index is between `0` and one less than the maximum element count, inclusive.

Return Value

A boolean value that is true if the element is defined; otherwise, it is false.

getElem()

Determines the value of an element.

Syntax

```
public
netscape.bsp.dataobject.IBSPDataObject getElem(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

If the returned value is of a primitive data type, it is copied into a primitive data object and the primitive data object is returned; otherwise, a reference to the data object is returned. The first index position in the list is `0`.

Return Value
An `IBSPDataObject` object that contains the value.

getElemBinary()
Determines the value of a byte array element.

Syntax
```
public byte[] getElemBinary(int index);
```

index – The position of the element whose value the application wants to determine.

Usage
The first index position in the list is `0`.

Return Value
A byte array that contains the value.

getElemBinarySize()
Determines the size of a byte array element.

Syntax
```
public byte[] getElemBinaryize(int index);
```

index – The position of the element whose size the application wants to determine.

Usage
The first index position in the list is `0`.

Return Value
An integer that contains the size, in bytes.

getElemCount()
Determines the current number of elements in the list.

Syntax
```
public int getElemCount();
```

Return Value
An integer that contains the number of elements.

getElemDataObject()

Determines the data object stored in an element.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject getElemDataObject
       (int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

A double precision floating point number that contains the value.

getElemDouble()

Determines the value of a double precision floating point number element.

Syntax

```
public double getElemDouble(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

A double precision floating point number that contains the value.

getElemFloat()

Determines the value of a floating point number element.

Syntax

```
public float getElemFloat(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

A floating point number that contains the value.

getElemFString()

Determines the value of a fixed length string element.

Syntax

```
public java.lang.String getElemFString(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

A string that contains the value.

getElemFStringMaxLen()

Determines the maximum length of a fixed length string element.

Syntax

```
public int getElemStringMaxLen(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains the size, in bytes.

getElemInt()

Determines the value of an integer element.

Syntax

```
public int getElemInt(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains the value.

getElemString()

Determines the value of a string element.

Syntax

```
public java.lang.String getElemString(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

A string that contains the value.

getElemType()

Determines the type of an element.

Syntax

```
public int getElemType(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

An integer value that specifies the type, which may be one of the following types:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2

Constant	Value
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

getElemVBinary()

Determines the value of a variable size byte array element.

Syntax

```
public byte[] getElemVBinary(int index);
```

index – The position of the element whose value the application wants to determine.

Variable Size Usage

The first index position in the list is `0`.

Return Value

A byte array that contains the value.

getElemVBinarySize()

Determines the size of a variable byte array element.

Syntax

```
public int getElemVBinarySize(int index);
```

index – The position of the element whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains the size, in bytes.

getMaxElemCount()

Determines the maximum number of elements in the list.

Syntax

```
public int getMaxElemCount();
```

Return Value

An integer that contains the number of elements.

isElemType()

Compares the element with the specified type.

Syntax

```
public boolean isElemType(int index,int typeNumber);
```

index – The position of the element whose value the application wants to determine.

typeNumber – The type with which the application wants to compare this element.

Usage

The first index position in the list is `0`. The type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

The application should always use a constant instead of a value in its Servlet, JSP, or EJB.

Return Value

A boolean value that is true if the type of the element matches the specified type; otherwise, it is false.

removeElem()

Removes an element in the list.

Syntax

```
public int removeElem(int index);
```

index – The position of the element to remove.

Usage

Elements after the element that was removed are moved down by one position in the list; for example, if the application removes `element 0`, all elements are shifted down by one such that `element 1` becomes `element 0`, `element 2` becomes `element 1`, and so on. The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemBinary()

Sets the value of a binary number element.

Syntax

```
public int setElemBinary(int index, byte[] buf);
```

index – The position of the element whose value the application wants to set.

buf – The value with which to set the object.

Usage

The first index position in the list is 0.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemDataObject()

Sets the data object stored in an element.

Syntax

```
public int setElemDataObject(int index,
       netscape.bsp.dataobject.IBSPDataObject value);
```

index – The position of the element whose value the application wants to set.

value – The value with which to set the element.

Usage

A value for a primitive data type is unwrapped and copied when it is set in the object; in other words, the value is stored in the object, not an object that contains the value. Otherwise, the data object itself is set to the value. The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemDataObjectList()

Sets an element to a new empty list.

Syntax

```
public int setElemDataObjectList(int index);
```

index – The position of the element to set.

Usage

An empty list is created at the specified index. The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemDataObjectStructure()

Sets an element to a new empty structure.

Syntax

```
public int setElemDataObjectStructure(int index);
```

index – The position of the element to set.

Usage
An empty structure is created at the specified index. The first index position in the list is `0`.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemDouble()
Sets the value of a double precision floating point number element.

Syntax
```
public int setElemDouble(int index, double value);
```

index – The position of the element whose value the application wants to set.

value – The value with which to set the element.

Usage
The first index position in the list is `0`.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemFloat()
Sets the value of a floating point number element.

Syntax
```
public int setElemFloat(int index, float value);
```

index – The position of the element whose value the application wants to set.

value – The value with which to set the element.

Usage
The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemFString()

Sets the value of a fixed length string element.

Syntax

```
public
int setElemFString(int index, java.lang.String stringValue);
```

index – The position of the element whose value the application wants to set.

stringValue – The value with which to set the element.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemInt()

Sets the value of an integer element.

Syntax

```
public int setElemInt(int index, int intValue);
```

index – The position of the element whose value the application wants to set.

intValue – The value with which to set the element.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemString()

Sets the value of a string element.

Syntax

```
public
int setElemString(int index, java.lang.String stringValue);
```

index – The position of the element whose value the application wants to set.

stringValue – The value with which to set the element.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setElemVBinary()

Sets the value of a variable size byte array element.

Syntax

```
public int setElemBinary(int index, byte[] buf);
```

index – The position of the element whose value the application wants to set.

buf – The value with which to set the element.

Usage

The first index position in the list is `0`.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setMaxElemCount()

Specifies the maximum number of elements in the list.

Syntax

```
public int setMaxElemCount(int maxElemCount);
```

maxElemCount – The maximum number of elements the application wants to specify.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectArray Interface

The `IBSPDataObjectArray` interface extends the `IBSPDataObjectList` interface to provide methods for accessing homogeneous data objects; an array's elements are homogeneous. An `IBSPDataObjectInfo` object is associated with an array to define the type of objects that the array contains.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`addElem()`	Add an element to an array.
`getElemTypeInfo()`	Determine the type information associated with an array.

addElem()

Add an element to an array.

Syntax

```
public int addElem();
```

Usage

The method creates an instance of the element type info object and adds it to the array. The data type of the element added is determined by the type information object associated with this array.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

getElemTypeInfo()

Determine the type information associated with an array.

Syntax

```
public
netscape.bsp.dataobject.IBSPDataObjectInfo getElemTypeInfo();
```

Return Value

An `IBSPDataObjectInfo` object that contains the type information.

IBSPDataObjectStructure Interface

The `IBSPDataObjectStructure` interface defines methods for accessing fields in a structure.

Package

`netscape.bsp.dataobject`

Access Methods

Method	Description
`getField()`	Determines the value of a field.
`getFieldBinary()`	Determines the value of a byte array field.
`getFieldDouble()`	Determines the value of a double precision floating point number field.
`getFieldFloat()`	Determines the value of a floating point number field.
`getFieldFString()`	Determines the value of a fixed length string field.
`getFieldDataObject()`	Determines the data object stored in an field.
`getFieldInt()`	Determines the value of an integer field.
`getFieldString()`	Determines the value of a string field.
`getFieldVBinary()`	Determines the value of a variable size byte array field.
`setFieldBinary()`	Sets the value of a binary number field.
`setFieldDataObject()`	Sets the data object stored in an field.
`setFieldDataObjectList()`	Sets an element to a new empty list.
`setFieldDataObjectStructure()`	Sets an element to a new empty structure.
`setFieldDouble()`	Sets the value of a double precision floating point number. field.
`setFieldFloat()`	Sets the value of a floating point number field.
`setFieldFString()`	Sets the value of a fixed length string field.
`setFieldInt()`	Sets the value of an integer field.
`setFieldString()`	Sets the value of a string field.
`setFieldVBinary()`	Sets the value of a variable size byte array field.

Other Methods

Method	Description
`fieldExists()`	Determines whether a field exists in the instance.
`fieldIsDefined()`	Determines whether a field is defined in the repository.
`getFieldBinarySize()`	Determines the size of a byte array field.
`getFieldCount()`	Determines the current number of fields in the structure.
`getFieldFStringMaxLen()`	Determines the maximum length of a fixed length string field.

Method	Description
`getFieldType()`	Determines the type of a field.
`getFieldVBinarySize()`	Determines the size of a variable byte array field.
`isFieldType()`	Compares the field type with the specified type.
`removeField()`	Removes a field.

fieldExists()

Determines whether a field exists in the instance.

Syntax

```
public boolean fieldExists(java.lang.String fieldName);
```

fieldName – The name of the field to test.

Usage

A field exists if its value exists in the object.

Return Value

A boolean value that is true if the field exists; otherwise, it is false.

fieldIsDefined()

Determines whether a field is defined in the repository.

Syntax

```
public boolean fieldIsDefined(java.lang.String fieldName);
```

fieldName – The name of the field to test.

Usage

A field is defined if its type information is defined in the repository.

Return Value

A boolean value that is true if the field is defined; otherwise, it is false.

getField()

Determines the value of a field.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject getField(
       java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Usage

If the returned value is of a primitive data type, it is copied into a new primitive data object and the primitive data object is returned; otherwise, a reference to the data object is returned.

Return Value

An `IBSPDataObject` object that contains the value.

getFieldBinary()

Determines the value of a byte array field.

Syntax

```
public byte[] getFieldBinary(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

A byte array that contains the value.

getFieldBinarySize()

Determine the size of a byte array field.

Syntax

```
public int getFieldBinarySize(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

An integer that contains the size, in bytes.

getFieldCount()

Determines the current number of fields in the structure.

Syntax

```
public int getFieldCount();
```

Return Value

An integer that contains the number of fields.

getFieldDataObject()

Determines the data object stored in an field.

Syntax

```
public
netscape.bsp.dataobject.IBSPDataObject getFieldDataObject(
      java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

An `IBSPDataObject` object.

getFieldDouble()

Determines the value of a double precision floating point number field.

Syntax

```
public double getFieldDouble(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

A double precision floating point number that contains the value.

getFieldFloat()

Determines the value of a floating point number field.

Syntax

```
public float getFieldFloat(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

A floating point number that contains the value.

getFieldFString()

Determines the value of a fixed length string field.

Syntax

```
public java.lang.String getFieldFString(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value
A string that contains the value.

getFieldFStringMaxLen()

Determines the maximum length of a fixed length string field.

Syntax

```
public int getFieldFStringMaxLen(java.lang.String fieldName);
```

fieldName – The name of the field whose size the application wants to determine.

Return Value
An integer that contains the size, in bytes.

getFieldInt()

Determines the value of an integer field.

Syntax

```
public int getFieldInt(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value
An integer that contains the value.

getFieldString()

Determines the value of a string field.

Syntax

```
public
java.lang.String getFieldString(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value
A string that contains the value.

getFieldType()

Determines the type of a field.

Syntax

```
public int getFieldType(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Usage

The first index position in the list is `0`.

Return Value

An integer value that specifies the type, which may be one of the following types:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

getFieldVBinary()

Determines the value of a variable size byte array field.

Syntax

```
public byte[] getFieldVBinary(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

A byte array that contains the value.

getFieldVBinarySize()

Determines the size of a variablebyte array field.

Syntax

```
public int getFieldVBinarySize(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to determine.

Return Value

An integer that contains the size, in bytes.

isFieldType()

Compares the field type with the specified type.

Syntax

```
public boolean isFieldType(java.lang.String fieldName,
       int typeNumber);
```

fieldName – The name of the field whose value the application wants to determine.

typeNumber – The type with which the application wants to compare this element.

Usage

The first index position in the list is `0`. The type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

The application should always use a constant instead of a value in its Servlet, JSP, or EJB; values are subject to change.

Return Value

A boolean value that is true if the type of the element matches the specified type; otherwise, it is false.

removeField()

Removes a field.

Syntax

```
public int removeField(java.lang.String fieldName);
```

fieldName – The name of the field the application wants to remove.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldBinary()

Sets the value of a binary number field.

Syntax

```
public
int setFieldBinary(java.lang.String fieldName, byte[] buf);
```

fieldName – The name of the field whose value the application wants to set.

buf – The value with which to set the field.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldDataObject()

Sets the data object stored in an field.

Syntax

```
public int setFieldDataObject(java.lang.String fieldName,
       netscape.bsp.dataobject.IBSPDataObject value;
```

fieldName – The name of the field whose value the application wants to set.

value – The value with which to set the field.

Usage

A value for a primitive data type is unwrapped and copied when it is set in the object; in other words, the value is stored in the object, not an object that contains the value. Otherwise, the data object itself is set to the value.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldDataObjectList()

Sets an element to a new empty list.

Syntax

```
public int setFieldDataObjectList(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to set.

Usage

An empty list is created in the specified field.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldDataObjectStructure()

Sets an element to a new empty structure.

Syntax

```
public
int setFieldDataObjectStructure(java.lang.String fieldName);
```

fieldName – The name of the field whose value the application wants to set.

Usage
An empty structure is created in the specified field.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldDouble()
Sets the value of a double precision floating point number. field.

Syntax
```
public
int setFieldDouble(java.lang.String fieldName, double value);
```

fieldName – The name of the field whose value the application wants to set.

value – The value with which to set the field.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldFloat()
Sets the value of a floating point number field.

Syntax
```
public
int setFieldFloat(java.lang.String fieldName, float value);
```

fieldName – The name of the field whose value the application wants to set.

value – The value with which to set the field.

Return Value
An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldFString()

Sets the value of a fixed length string field.

Syntax

```
public
int setFieldFString(java.lang.String fieldName,
       java.lang.String stringValue);
```

fieldName – The name of the field whose value the application wants to set.

stringValue – The value with which to set the field.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldInt()

Sets the value of an integer field.

Syntax

```
public
int setFieldInt(java.lang.String fieldName, int intValue);
```

fieldName – The name of the field whose value the application wants to set.

intValue – The value with which to set the field.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldString()

Sets the value of a string field.

Syntax

```
public int setFieldString( java.lang.String fieldName,
       java.lang.String stringValue);
```

fieldName – The name of the field whose value the application wants to set.

stringValue – The value with which to set the field.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setFieldVBinary()

Sets the value of a variable size byte array field.

Syntax

```
public
int setFieldBinary(java.lang.String fieldName, byte[] buf);
```

fieldName – The name of the field whose value the application wants to set.

buf – The value with which to set the field.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectStructureInfo Interface

The `IBSPDataObjectStructureInfo` interface defines methods for obtaining information about a `IBSPDataObjectStructure` object.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`definesField()`	Determines if a field is defined in the structure information.
`getFieldInfo()`	Determines the field information for a field in a structure object.
`getFields()`	Determines which fields are in a structure object.

definesField()

Determines if a field is defined in the structure information.

Syntax

```
public boolean definesField(java.lang.String fieldName);
```

fieldName – The name of the field to test.

Return Value

A boolean value that is true if the field is defined; otherwise, it is false.

getFieldInfo()

Determines the field information for a field in a structure object.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObjectInfo getFieldInfo(
       java.lang.String fieldName);
```

fieldName – The name of the field whose information the application wants to obtain.

Return Value

A `IBSPDataObject` object the contains the field information.

getFields()

Determines which fields are in a structure object.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObjectStructure getFields);
```

Return Value

A `IBSPDataObjectStructure` object the contains the fields.

IBSPDataObjectInfo Interface

The `IBSPDataObjectInfo` interface defines a method for creating an instance of a data object of the same type as the interface.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`createInstance()`	Creates an instance of the `IBSPDataObjectInfo` object.

createInstance()

Creates an instance of the `IBSPDataObjectInfo` object.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject createInstance();
```

Return Value

An instance of an `IBSPDataObject` object.

IBSPDataObjectListInfo Interface

The `IBSPDataObjectListInfo` interface defines methods for determining or specifying the initial size and maximum size of a list object.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`getInitialSize()`	Determines the initial capacity of the list.
`getMaxElemCount()`	Determines the maximum number of elements.
`setInitialSize()`	Specifies the initial capacity of the list.
`setMaxElemCount()`	Specifies the maximum number of elements.

getInitialSize()

Determines the initial capacity of the list.

Syntax

```
public int getInitialSize();
```

Return Value

An integer that contains the initial size of the list.

getMaxElemCount()

Determines the maximum number of elements.

Syntax

```
public int getInitialSize();
```

Return Value

An integer that contains the maximum size of the list.

setInitialSize()

Specifies the initial capacity of the list.

Syntax

```
public int setInitialSize(int intialSize);
```

initialSize – The initial capacity of the list.

Usage

The list does not physically grow until the capacity has been exceeded.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setMaxElemCount()

Specifies the maximum number of elements.

Syntax

```
public int setMaxElemCount(int maxElemCount);
```

maxElemCount – The maximum size of the list.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectArrayInfo Interface

The `IBSPDataObjectArrayInfo` interface extends the `IBSPDataObjectListInfo` interface to define methods for getting and setting the type of elements in the array and the info object associated with the array.

Package

`netscape.bsp.dataobject`

Methods

Method	Description
`getArrayElemType()`	Determines the array's element type.
`getArrayElemTypeInfo()`	Retrieves the array's type info object.
`setArrayElemType()`	Specifies the array's element type.
`setArrayElemTypeInfo()`	Sets the array's type info object.

getArrayElemType()

Determines the array's element type.

Syntax

```
public int getArrayElemType();
```

Usage

Use this method if the type is primitive; otherwise, use `setArrayElemTypeInfo()`. The element type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

The application should always use a constant instead of a value in its Servlet, JSP, or EJB.

Return Value

An integer that contains the type number.

getArrayElemTypeInfo()

Retrieves the array's type info object.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObjectInfo
       getArrayElemTypeInfo();
```

Return Value

An `IBSPDataObjectInfo` object.

setArrayElemType()

Specifies the array's element type.

Syntax

```
public int setArrayElemType(int elemTypeNumber);
```

elemTypeNumber – The type of elements in the array.

Usage

Use this method if the type is primitive; otherwise, use `setArrayElemTypeInfo()`. The element type may be one of the following:

Constant	Value
`TN_BINARY.TN_BINARY`	`5`
`TN_DATAOBJECT.TN_DATAOBJECT`	`128`
`TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY`	`161`
`TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO`	`211`
`TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST`	`160`
`TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO`	`210`
`TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO`	`209`
`TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO`	`212`
`TN_DOUBLE.TN_DOUBLE`	`3`
`TN_FLOAT.TN_FLOAT`	`2`
`TN_FSTRING.TN_FSTRING`	`6`
`TN_STRING.TN_STRING`	`4`
`TN_UNDEFINED.TN_UNDEFINED`	`0`
`TN_VBINARY.TN_VBINARY`	`7`

The application should always use a constant instead of a value in its Servlet, JSP, or EJB.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setArrayElemTypeInfo()

Sets the array's type info object.

Syntax

```
public int setArrayElemTypeInfo
       (netscape.bsp.dataobject.IBSPDataObjectInfo elemTypeInfo);
```

elemTypeInfo – The type info object to associate with this array.

Usage

Use this method if the type is not primitive; otherwise, use `setArrayElemType()`. This method calls `setArrayElemType()` to set the element type.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

IBSPDataObjectDataItr Interface

The `IBSPDataObjectDataItr` interface defines methods for iterating over objects in a list, array, or structure. The application calls the `next()` method to move to the next position in the list, array, or structure. The application calls the `getCurrentType()` method to determine the type of object at the current location.

Package

`netscape.bsp.dataobject`

Accessor Methods

Method	Description
`getCurrent()`	Determines the value of the attribute at the current position.
`getCurrentBinary()`	Determines the value of a byte array attribute at the current position.
`getCurrentDataObject()`	Determines the value of a data object attribute at the current position.
`getCurrentDouble()`	Determines the value of a double precision floating point number attribute at the current position.
`getCurrentFloat()`	Determines the value of a floating point number attribute at the current position.
`getCurrentFString()`	Determines the value of a fixed length string attribute at the current position.
`getCurrentInt()`	Determines the value of an integer attribute at the current position.
`getCurrentString()`	Determines the value of a string attribute at the current position.
`getCurrentVBinary()`	Determines the value of a variable length byte array attribute at the current position.
`setCurrentBinary()`	Specifies the value of a byte array attribute at the current position.
`setCurrentDataObject()`	Specifies the value of a data object attribute at the current position.
`setCurrentDouble()`	Specifies the value of a double precision floating point number attribute at the current position.
`setCurrentFloat()`	Specifies the value of a floating point number attribute at the current position.
`setCurrentFString()`	Specifies the value of a fixed length string attribute at the current position.
`setCurrentInt()`	Specifies the value of an integer attribute at the current position.
`setCurrentString()`	Specifies the value of a string attribute at the current position.
`setCurrentVBinary()`	Specifies the value of a variable length byte array attribute at the current position.

Other Methods

Method	Description
`getCurrentBinarySize()`	Determines the size of the byte array attribute at the current position.
`getCurrentFieldName()`	Determines the field name of the field at the current position.
`getCurrentType()`	Determines the type of the attribute at the current position.
`getCurrentVBinarySize()`	Determines the size of the variable length byte array attribute at the current position.
`isCurrentType()`	Compares the type of the attribute at the current position with the specified type.
`next()`	Advances the current position to the next attribute.

getCurrent()

Determines the value of the attribute at the current position.

Syntax

```
public netscape.bsp.dataobject.IBSPDataObject getCurrent();
```

Usage

If the returned value is of a primitive data type, it is copied into a new primitive data object and the primitive data object is returned; otherwise, a reference to the data object is returned.

Return Value

An `IBSPDataObject` object that contains the value at the current position.

getCurrentBinary()

Determines the value of a byte array attribute at the current position.

Syntax

```
public byte[] getCurrentBinary();
```

Return Value

A byte array that contains the value.

getCurrentBinarySize()

Determines the size of the byte array attribute at the current position.

Syntax
```
public int getCurrentBinarySize();
```

Return Value
An integer that contains the size, in bytes.

getCurrentDataObject()
Determines the value of a data object attribute at the current position.

Syntax
```
public netscape.bsp.dataobject.IBSPDataObject
       getCurrentDataObject();
```

Return Value
An `IBSPDataObject` object.

getCurrentDouble()
Determines the value of a double precision floating point number attribute at the current position.

Syntax
```
public double getCurrentDouble();
```

Return Value
A double precision floating point number that contains the value.

getCurrentFieldName()
Determines the field name of the field at the current position.

Syntax
```
public java.lang.String getCurrentFieldName();
```

Usage
When iterating over a structure, the field name is returned. When iterating over a list or array, the element number is returned as a string enclosed in brackets; for example, `[0]` is returned for the first element.

Return Value
An string that contains the name.

getCurrentFloat()

Determines the value of a floating point number attribute at the current position.

Syntax

```
public float getCurrentFloat();
```

Return Value

A floating point number that contains the value.

getCurrentFString()

Determines the value of a fixed length string attribute at the current position.

Syntax

```
public java.lang.String getCurrentFString();
```

Return Value

A string that contains the value.

getCurrentInt()

Determines the value of an integer attribute at the current position.

Syntax

```
public int getCurrentInt();
```

Return Value

An integer that contains the value.

getCurrentString()

Determines the value of a string attribute at the current position.

Syntax

```
public java.lang.String getCurrentString();
```

Return Value

A string that contains the value.

getCurrentType()

Determines the type of the attribute at the current position.

Syntax
```
public int getCurrentType();
```

Return Value
An integer value that specifies the type. The type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

getCurrentVBinary()
Determines the value of a variable length byte array attribute at the current position.

Syntax
```
public byte[] getCurrentBinary();
```

Return Value
A byte array that contains the value.

getCurrentVBinarySize()
Determines the size of the variable length byte array attribute at the current position.

Syntax
```
public int getCurrentVBinarySize();
```

Return Value

An integer that contains the size, in bytes.

isCurrentType()

Compares the type of the attribute at the current position with the specified type.

Syntax

```
public boolean isCurrentType(int typeNumber);
```

TypeNumber – The type with which the application wants to compare the attribute at this position.

Usage

The type may be one of the following:

Constant	Value
TN_BINARY.TN_BINARY	5
TN_DATAOBJECT.TN_DATAOBJECT	128
TN_DATAOBJECT_ARRAY.TN_DATAOBJECT_ARRAY	161
TN_DATAOBJECT_ARRAY_INFO.TN_DATAOBJECT_ARRAY_INFO	211
TN_DATAOBJECT_LIST.TN_DATAOBJECT_LIST	160
TN_DATAOBJECT_LIST_INFO.TN_DATAOBJECT_LIST_INFO	210
TN_DATAOBJECT_PRIM_INFO.TN_DATAOBJECT_PRIM_INFO	209
TN_DATAOBJECT_STRUCT_INFO.TN_DATAOBJECT_STRUCT_INFO	212
TN_DOUBLE.TN_DOUBLE	3
TN_FLOAT.TN_FLOAT	2
TN_FSTRING.TN_FSTRING	6
TN_STRING.TN_STRING	4
TN_UNDEFINED.TN_UNDEFINED	0
TN_VBINARY.TN_VBINARY	7

The application should always use a constant instead of a value in its Servlet, JSP, or EJB.

Return Value

A boolean value that is true if the type of the attribute matches the specified type; otherwise, it is false.

next()

Advances the current position to the next attribute.

Syntax

```
public boolean next();
```

Usage

Use the `next()` method to advance the current position to the next one in the list or structure. After the application calls `next()` the first time, the current position becomes `0`.

Return Value

A boolean value that is true if there are more attributes in the list or structure; it is false if the current position is the last position in the list or structure.

setCurrentBinary()

Specifies the value of a byte array attribute at the current position.

Syntax

```
public int setCurrentBinary(byte[] buf);
```

buf – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentDataObject()

Specifies the value of a data object attribute at the current position.

Syntax

```
public int setCurrentDataObject(
       netscape.bsp.dataobject.IBSPDataObject value);
```

value – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentDouble()

Specifies the value of a double precision floating point number attribute at the current position.

Syntax

```
public int setCurrentDouble(double value);
```

value – The double precision floating point number with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentFloat()

Specifies the value of a floating point number attribute at the current position.

Syntax

```
public int setCurrentFloat(float value);
```

value – The floating point number with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentFString()

Specifies the value of a fixed length string attribute at the current position.

Syntax

```
public int setCurrentFString(java.lang.String value);
```

value – The string value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentInt()

Specifies the value of an integer attribute at the current position.

Syntax

```
public int setCurrentInt(int value);
```

value – The floating point number with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentString()

Specifies the value of a string attribute at the current position.

Syntax

```
public int setCurrentString(java.lang.String value);
```

value – The string value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

setCurrentVBinary()

Specifies the value of a string attribute at the current position.

Syntax

```
public int setCurrentVBinary(byte[] buf);
```

buf – The value with which to set the attribute.

Return Value

An integer that contains:

`GXE.SUCCESS` - if the operation is successful

`GXE.FAIL` - if the operation fails

BspException Class

The `netscape.bsp.BspException` exception extends the `java.lang.RuntimeException`. All exceptions thrown in UIF and the enterprise connector extend the `netscape.bsp.BspException` class.

Package

`netscape.bsp`

Methods

Method	Description
`getInfo()`	Gets the exception details information on the Exception

getInfo()

Gets the exception information details.

Syntax

```
public IBSPDataObjectStructure getInfo();
```

Return Value

`IBSPDataObjectStructure` object containing the exception information.

DataObjectException Class

The `DataObjectException` exception is thrown when a data object exception occurs in the UIF runtime. `DataObjectException` extends `netscape.bsp.BspException`. Note that all data object exceptions extend the `DataObjectException` interface.

Package

`netscape.bsp.dataobject.exceptions`

FieldNameIsNullException Class

The `FieldNameIsNullException` is thrown when a data object field name being accesed is null. Please check the UIF Repository contents to determine the availability of the field being accessed and make sure a name is assigned to a field.

Package

`netscape.bsp.dataobject.exceptions`

FieldNameTooLongException Class

The `FieldNameTooLongException` exception is thrown when a data object field is being accessed whose name is too long. The maximum field length allowed is 64. Please check the UIF repository content and modify the name accordingly.

Package

`netscape.bsp.dataobject.exceptions`

FieldNotFoundException Class

The `FieldNotFoundException` exception is thrown when a data object field or the data object is not found in the repository. Check the UIF Repository content and make sure the field being accessed exist in the UIF Repository.

Package

`netscape.bsp.dataobject.exceptions`

IllegalPathException Class

The `IllegalPathException` exception is thrown when a data object or data object field name is specified wrong. Please verify the application against the UIF Repository contents and modify the application accordingly.

Package

`netscape.bsp.dataobject.exceptions`

IllegalRecusionException Class

The `IllegalRecusionException` exception is thrown when data objects are being accessed in the wrong manner. Please refer to Chapter 5, “Programming Model” for more details on the illegal data object recursions.

Package

`netscape.bsp.dataobject.exceptions`

IndexOutOfBoundsException Class

The `IndexOutOfBoundsException` exception is thrown when the application tries to traverse a non existing index.

Package

`netscape.bsp.dataobject.exceptions`

MethodUnsupportedException Class

The `MethodUnsupportedException` exception is thrown when the application tries to access a method that is unsupported on a given data object.

Package

`netscape.bsp.dataobject.exceptions`

TypeMismatchException Class

The `TypeMismatchException` exception is thrown when a type mismatch in the data object values are being set by the application. Please check the data type in the UIF Repository and set the values accordingly.

Package

`netscape.bsp.dataobject.exceptions`

Glossary

Adapter See Enterprise Connector.

Application a J2EE Application running inside an iPlanet Application Server.

Application Programming Interface (API) The functions that the application program uses to perform tasks. For example, managing files, accepting text or numbers in a document, and displaying information on the screen. The API defines a standard way of using features.

Application Server A comprehensive solution that enables enterprises to easily develop, deploy, and manage business critical Internet applications.

Bind Duration is how long a connection is bound to the client J2EE component.

Bind Duration Escalation to escalate the connection from `method-bound` to `sp-bound` and vice versa by the enterprise connector.

Bind Duration Timeout is to forcibly get the connection from the service provider back to the pool, so that other service providers could use it.

BSP See UIF.

Configuration Info Enterprise connector specific EIS configuration information. This information includes access details like EIS host names, port numbers, languages, load balancing features, etc. Furthermore, the configuration information is loosely defined to include pooling configuration definitions, user mapping definitions, etc.

Data Block a data object which encapsulates all inputs and outputs of this operation's execution. This object can have default or initial values. The enterprise connector is responsible for marshalling and unmarshalling the data object contents, to and from backend formats.

Data Mining A service provided by the enterprise connector provider in a GUI-based tool form, in order to mine the service and data object definition information from the EIS systems and export it to the repository. For example, in case of the R/3 enterprise connector, the R/3 data mining tool fetches the BAPI information and In and Out parameters and populates the repository.

Data Object a backend system independent data representation. It also provides rich and consistent set of introspect, access and mutation APIs for developers to manipulate data objects. Data objects shield the user from learning complicate legacy system data formats. It is the only common data format that is used to communicate among UIF and enterprise connector extensions. Each enterprise connector extension is required to understand a data object and translate native data formats to this common form and vice versa.

Data Object Array has elements of homogeneous types addressed by an integer index.

Data Object Collection is the super interface of data object structure, list and array.

Data Object List has elements of heterogeneous types addressed by an integer index. A data object list has a special count attribute to hold the field count.

Data Object Primitive a data object that wraps a primitive value around the data object. For example, an integer, a string, binary buffer, etc.

Data Object Structure is fields of heterogeneous types addressed by either a name or an integer index. Two data object structure fields cannot have the same name. A data object structure has a special count attribute to hold the field count.

Datasource Data source is a logical representation of the EIS instance inside UIF. The definition of a datasource is stored inside the repository. This definition contains EIS specific service provider definitions, pool definitions, data type definitions, function object definitions and user mapping definitions.

Data Object Type Definitions the structural information of the Data Objects, mined by the Administrator to be made available in the Repository.

Deployment The process whereby software is installed into an operational environment.

Deployment Descriptor An XML file provided with each module and application that describes how they should be deployed. The deployment descriptor directs a deployment tool to deploy a module or application with specific container options and describes specific configuration requirements that a deployer must resolve.

Distributed Application An application made up of distinct components running in separate runtime environments, usually on different platforms connected via a network. Typical distributed applications are two-tier (client-server), three-tier (client-middleware-server), and multiplier (client-multiple middleware-multiple servers).

Document Type Definition (DTD) Is a file (or several files) written in XMLs declaration syntax, which contain a formal description of a particular type of document. It describes the names that can be used for element types, where they may occur and how they fit together.

Enterprise Connector EIS specific extension that implements the UIF enterprise connector contract, and provides marshalling and unmarshalling of UIF data types to EIS specific data types. The enterprise connector also provides rich exceptions which incorporate backend specific error codes to the application logic.

Enterprise Information System (EIS) EIS can be interpreted as packaged enterprise application, transaction systems or user applications. For example, R/3, PeopleSoft, Tuxedo, and CICS.

Enterprise Server Beans (EJB) Is a *non visual* software component running on the server part of an application. An EJB is usually configured at deployment time by editing its properties.

Extensible Markup Language (XML) Is the universal format used for structured documents and data on the web. XML is extensible because it is not a fixed format like HTML. XML is not a single, predefined markup language, it's a meta language – a language for describing other languages – which lets you design your own markup. A predefined markup language like HTML defines a way to describe information in one specific document class only. XML lets you define your own customized markup languages for different document classes. XML can do this because it's written in SGML, the international standard meta language for text markup systems.

Function Object is a logical group of operations. Each operation can be considered as a method with its own input and output parameters. The parameters are represented in a single data object named dataBlock. The operations are related because they share a common state machine. The state is maintained by runtime from call-to-call.

Function Object Definitions the structural information of the Function Objects, mined by the Administrator to be made available in the Repository.

Function Object Template defines an instance of a Function Object. All the Function Object instances of a specific template type follow the same structure.

Interaction Models a conversational model between the UIF and the enterprise connector. UIF currently supports only synchronous request/response interaction model.

J2EE application Any deployable unit of J2EE functionality. This can be a single module or a group of modules packaged into an `.ear` file with a J2EE application deployment descriptor. J2EE applications are typically engineered to be distributed across multiple computing tiers.

J2EE product An implementation that conforms to the J2EE platform specification.

Java Java is a network programming language invented by Sun Microsystems that is designed for writing programs that can be safely downloaded through the Internet and immediately run without fear of viruses or other harm to users or files.

Java 2 Enterprise Edition (J2EE) An environment for developing and deploying enterprise applications.

JavaServer Page (JSP) Allows developers to rapidly develop and easily maintain, information rich, dynamic web pages that leverage existing business systems. JSP also enables rapid development of web-based applications that are platform independent. Additionally, JSP is an extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to return dynamic content to a client. Typically the template data is HTML or XML elements, and in many cases the client is a Web browser.

Lightweight Directory Access Protocol (LDAP) Defines a network representation of a search filter transmitted to an LDAP server.

Marshalling and Unmarshalling it is defined as the conversions of UIF data type formats to EIS specific data formats. This is the primary responsibility of the enterprise connector.

Metadata Metadata can be loosely defined as the configuration, function object, data object, and service provider information available in the repository. The metadata service which lives as a core service in UIF is responsible for interpreting the metadata definitions and instantiate corresponding in-memory objects.

Pool Definitions the pooling information to be used by UIF, mined by the Administrator to be made available in the Repository.

Pooling Algorithm used to enable efficient use of the scarce resources.

Pooling Configuration the pooling configuration information used by the UIF, mined by the Administrator to be made available in the Repository.

Process Manager (PM) IPlanet process manager tool allows the user to defined process models and their interactions. Process manager's runtime engine interprets the process model definitions and executes the process items as per the definition.

Property Set a data object whose value settings target this operation's execution to a specific backend business function. For example, dbName, storedProc name, BAPI name, Peoplesoft message identifier, program name, queue name, etc. The settings also control the operation's execution.

Repository UIF repository is a persistent store that stores information needed by UIF components. For example, runtime, metadata service, tools – repository browser, connector management console, enterprise connectors, etc. The repository is a hierarchy of information where semantic categories (or types) of information stored in repository are – function objects, data objects, data info objects, enterprise connector (adapter) configuration details (service providers), pool configuration details, web user-to-EIS user mapping information, etc. An importer / exporter tool imports and exports repository contents to and from the user readable form (XML). UIF repository services provide cached runtime access to the information stored in the UIF repository. Repository APIs provides functionalities like – access, navigation and construction of data objects from the repository.

Repository Browser The repository browser allows you to view the repository contents and to perform operations such as browsing content and exporting and importing function object, data object, and configuration information to and from XML files.

Runtime Manager an object factory to construct data object, service provider and function object based on either type or template information. Further, the runtime manager encapsulates advance features such as connection pooling, thread hand off and connection bind escalation from a developer.

Server A computer or a software package, that provides a specific kind of service to client software running on other computers. The term can refer to a particular piece of software. For example, a iPlanet web server or iPlanet application server, or the machine that the software is running on, A single server machine could have several different server software packages running on it.

Service Provider A service provider is a logical representation of the connection to the backend datasource. The service provider definition is usually appropriately preconfigured (by the datasource administrator) to point to the EIS, as well as preconfigured with appropriate connection pooling and lifecycle management settings.

Service Provider Definitions The configuration information of the Service Provider, mined by the Administrator to be made available in the Repository.

Servlet An application or script written in Java and executed on a server. A servlet extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a request-response paradigm.

System Administrator The person responsible for configuring and administering the enterprise's computers, networks, and software systems.

Tools Are used for the development, packaging, and deployment of applications.

Transmission Control Protocol/Internet Protocol (TCP/IP) TA suite of protocols that defines the Internet. Designed for the UNIX operating system, but is now available for every major operating system.

Unified Integration Framework (UIF) UIF appears in the package names and repository browser. The UIF consists of a unified API suite for all enterprise connectors.

User Mapping Definitions These are definitions of user mapping tables from web domain to EIS domain for a specific EIS system. The contents of the user mapping tables are managed via the management console.

XML See Extensible Markup Language.

Index

E

F

G

I

J

L

M

N

P

R

S

U

V