Developer's Guide

iPlanet Application Server Enterprise Connector for R/3

Version 6.5

806-5508-02
August 2002

Contents

List of Figures

List of Procedures

List of Tables

List of Code Examples

Preface

The *iPlanet Application Server Enterprise Connector for R/3 Developer's Guide* gives a brief overview of the R/3 connector and explains how to write the servlet or Enterprise Java Bean (EJB) applications. The *iPlanet Application Server Enterprise Connector for R/3 Developer's Guide* is written for application programmers who develop internet or intranet applications for the R/3 back-end system.

This preface contains information about the following topics:

- Prerequisites
- What's in This Guide
- Documentation Conventions
- Online Guide
- Related Information

Prerequisites

This guide assumes that you are familiar with the following topics:

- iPlanet Application Server programming concepts
- The Internet and World Wide Web
- R/3 Programming Concepts
- Java Programming Language and Java Servlets
- Enterprise JavaBeans
- Java Programming Language

What's in This Guide

The *iPlanet Application Server Enterprise Connector for R/3 Developer's Guide* provides the information you need to know to write servlets or EJBs to connect the iPlanet Application Enterprise Connector for R/3 to both the Unified Integration Framework (UIF) and to your R/3 backend.

The following table lists a short summary of what each chapter covers.

See this chapter	If you want to do this
Chapter 1, "Overview"	Familiarize yourself with conceptual information before writing UIF APIs servlets or EJBs.
Chapter 2, "Viewing the Repository Contents"	View all contents of the Repository Browser.
Chapter 3, "Working With Data Objects"	Aquire Unified Integration Framework objects and execute function objects. The chapter also describes how to use the UIF API to develop a servlet or EJB which communicates with the back-end system.
Chapter 4, "Programming Samples"	View a simple sample application and view examples of code that can be used to set up the R/3 connector.
Appendix A, "Error Messages"	Display error messages for the iPlanet Application Server Enterprise Connector for R/3 and their codes. The chapter also includes a sample of the error handling code that can be used by the servlet developer.

Documentation Conventions

This guide uses URLs of the form:

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

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

This guide uses the following font conventions:

- The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, path names, directory names, and HTML tags.
- *Italic* type is used for book titles, emphasis, variables and placeholders, and words used in the literal sense.

Online Guide

You can find the *iPlanet Application Server Enterprise Connector for R/3 Developer's Guide* online in PDF and HTML formats. To locate these files, use the following URL:

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

Related Information

In addition to this guide, there is additional information available for administrators, end users and developers. The following lists these documents:

- *iPlanet Application Server Enterprise Connector for R/3 Administrator's Guide*
- *iPlanet Web Server Developer's Guide*
- *iPlanet Application Server Administrator's Guide*
- *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
- *Unified Integration Framework (UIF) Release Notes*
- *Unified Integration Framework (UIF) Developer's Guide*)

Chapter 1

Overview

The *iPlanet Application Server Enterprise Connector for R/3* is used for building and delivering scalable applications that integrate the application server with R/3 Enterprise Resource Planning (ERP) management systems. The *iPlanet Application Server Enterprise Connector for R/3* enables communication between an end user and a remote R/3 back-end system.

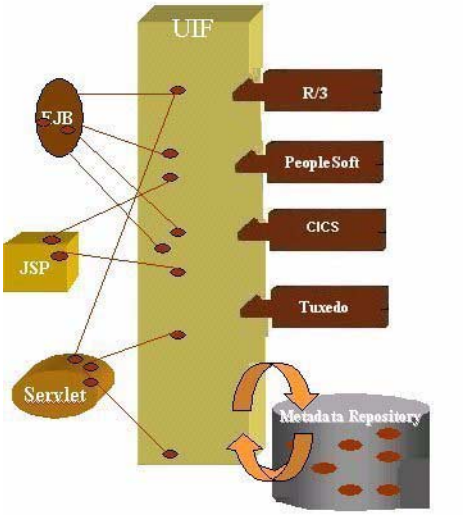
This chapter contains information about the following topics:

- Unified Integration Framework (UIF)
- Three-tier Application Model
- Enterprise Connector Tools for R/3

Unified Integration Framework (UIF)

The UIF is an application programming framework that provides a single Application Programming Interface (API) to access different back-end systems. A connector is developed for each back-end system to allow communication between the UIF API and the back-end system, as shown in Figure 1-1. The UIF API is the only API necessary to access the back-end system.

Figure 1-1 Unified Integration Framework



The UIF enables development of server extensions that integrate with legacy systems, client-server applications, and third-party Internet solutions. These extensions provide a consistent access layer to disparate back-end systems, dramatically reducing development effort. The framework provides support for features such as object-pooling, and distributed state and session management.

A generic data repository is also part of the UIF, which is used to hold metadata parameters and other information about the back-end system. For example, the metadata often describes the physical connection between systems, the data that is available, and methods you can use to process data.

The implementation of each connector is specific to its back-end Enterprise Information System (EIS) since each EIS is different.

UIF Services

The UIF is a component of the iPlanet Application Server. The iPlanet Application Server plays a prominent role in a three-tier application model. See the Three-tier Application Model for a description. The UIF mediates between the iPlanet Application Server application and the EIS (back-end) tier.

The UIF provides an API to access the following services:

- Runtime
- Data Object Services (available to users only)
- Repository and Metadata Services

Runtime

The UIF runtime services supply core services for resource management, thread management, communication and life cycle management, and exception management. The UIF runtime services understand and interpret metadata repository contents.

Data Object Services

The Data Object Services implement universal data representation common to all connectors. See Chapter 3, “Working With Data Objects” for a description of data objects.

Repository and Metadata Services

The UIF repository and metadata services model a persistent information hierarchy that supports datatype definitions and inheritance. They also manage the instances and reuse of data objects from datatype definitions.

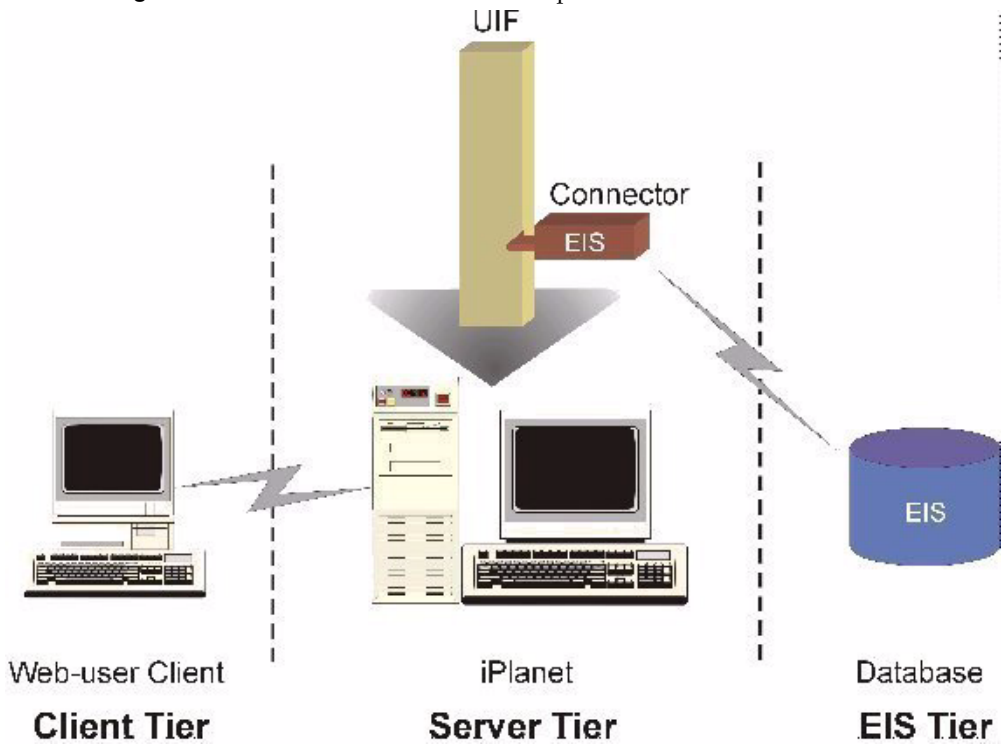
Three-tier Application Model

The machine and software involved are divided into three tiers:

- Client Tier
- Server Tier
- EIS (back-end) Tier

The connectors serve as an essential link allowing the server tier to communicate with the EIS tier, as shown in Figure 1-2. Communication between the application server and the back-end EIS is facilitated by the UIF API. This layer of functionality resides as an added layer to the iPlanet Application Server and enables data communication with diverse back-end EIS in a seamless and uniform manner.

Figure 1-2 Three-tier web-based Computer Model



Client Tier

The client tier is represented as the user interface. Requests for data originate here, represented by web browsers or rich clients (such as Java applets).

Server Tier

The server tier is represented by an application server, and optionally a web server such as the iPlanet Web Server Enterprise Edition. The server tier houses the business logic (your application servlets and/or Enterprise Java Beans), and provides scalability, high availability load balancing, and integration with a variety of data sources.

EIS (back-end) Tier

The back-end tier is represented by Enterprise Resource Planning (ERP) systems databases or other back-end data systems, such as R/3.

Enterprise Connector Tools for R/3

The Enterprise Connector Tools are as follows:

- Management Console - includes User Mapping and Data Mining Tools
  - User Mapping - allows you to map user IDs for access into the back-end system, and to edit and manage data sources.

    A data source represents the connection to the back-end system.
  - Data Mining Tool - includes capabilities such as determining the available functions in the back-end system, translating and reformatting data, and loading data into the data repository.
- Repository Browser - allows you to browse data in the repository. You can view the available functions (input and output parameters) for the back-end system. For developers use of the Repository Browser see Chapter 2.

These tools are thoroughly described in the *iPlanet Application Server Enterprise Connector for R/3 Administrator's Guide*.

Chapter 2

Viewing the Repository Contents

The Repository Browser is designed to provide the developer with a convenient tool that can be used to view the contents of the repository.

This chapter describes the following topics:

- Overview of the Repository Browser
- Viewing the Repository
- The Service Provider Object
- Function Objects
- Operations
- Entity Mapping

Overview of the Repository Browser

The main function of the Repository Browser is to view the repository contents. The developer must be able to see the contents of the repository in order to be able to program an application. Variable values in the repository can not be changed using the browser. XML files can be imported and exported using the browser.

CAUTION The Repository Browser should not be used for editing even though import, export, and delete actions on repository nodes are enabled. Only advanced administrators should use these functions.

To Launch the Repository Browser on Windows NT/2000

- Select Programs > iPlanet Application Server 6.5 > UIF 6.5 Repository Browser.

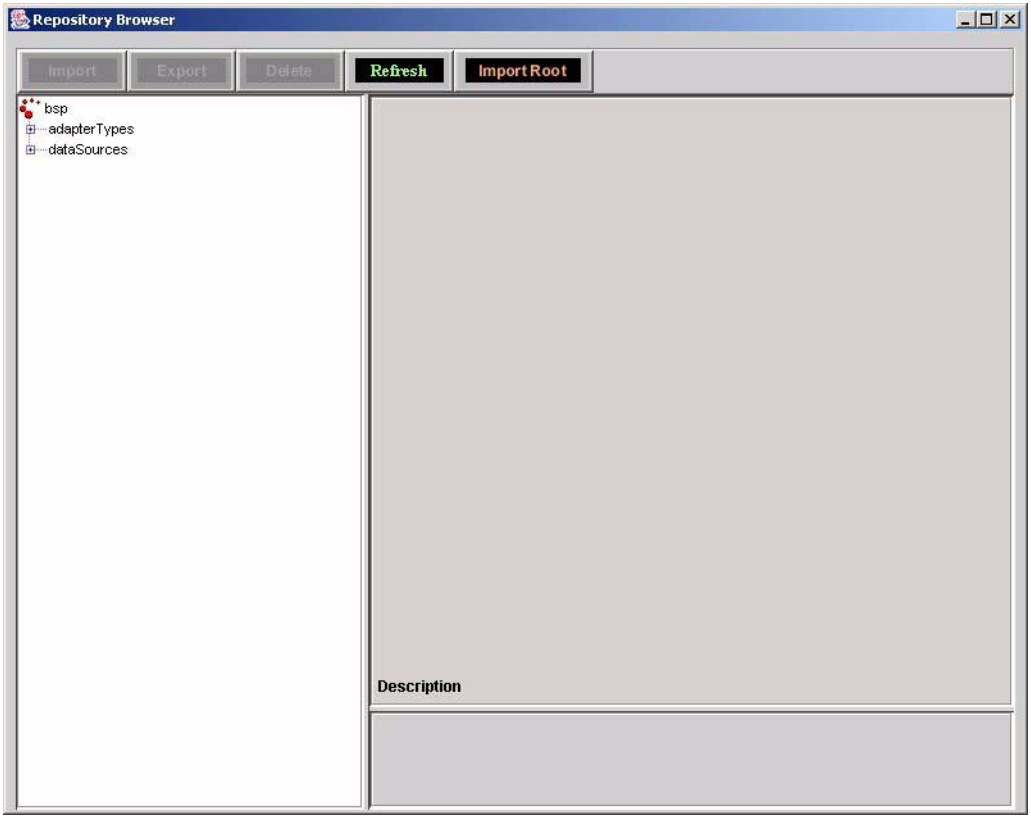
To Launch the Repository Browser on Solaris

- Run the following script:

  ```
  <ias directory>/APPS/bin/bspbrowser.sh
  ```

  The Repository Browser is shown in Figure 2-1.

Figure 2-1 Repository Browser



Viewing the Repository

The Repository Browser is divided into two panes. When you open the browser, the left pane displays nodes containing the adapter (connector) types and dataSources. These nodes are hierarchical and can be expanded to show details of the data structure and function objects. The right pane shows the properties of the node selected on the left side.

Viewing the Hierarchy

You can expand and collapse your view of the repository. Initially, the hierarchy displays the following:

- the root node
- connector types
- data sources

To Refresh the Display of the Repository Contents

- Click Refresh to refresh the display of the Repository contents.

Viewing Data Objects

The Repository Browser allows you to view data object templates, data object types, and data object image nodes in different ways. The node specifies the view that is currently displayed.

Details are included for the following objects:

- The Service Provider Object
- Function Objects
- Operations

The Service Provider Object

The Service Provider Object is the logical representation of a connection to a back-end system. Usually, the Service Provider Object is not bound to a physical connection until it is absolutely necessary. The Service Provider Object is below the service provider template, as shown in Figure 2-2.

Figure 2-2 Service Provider Configuration Object

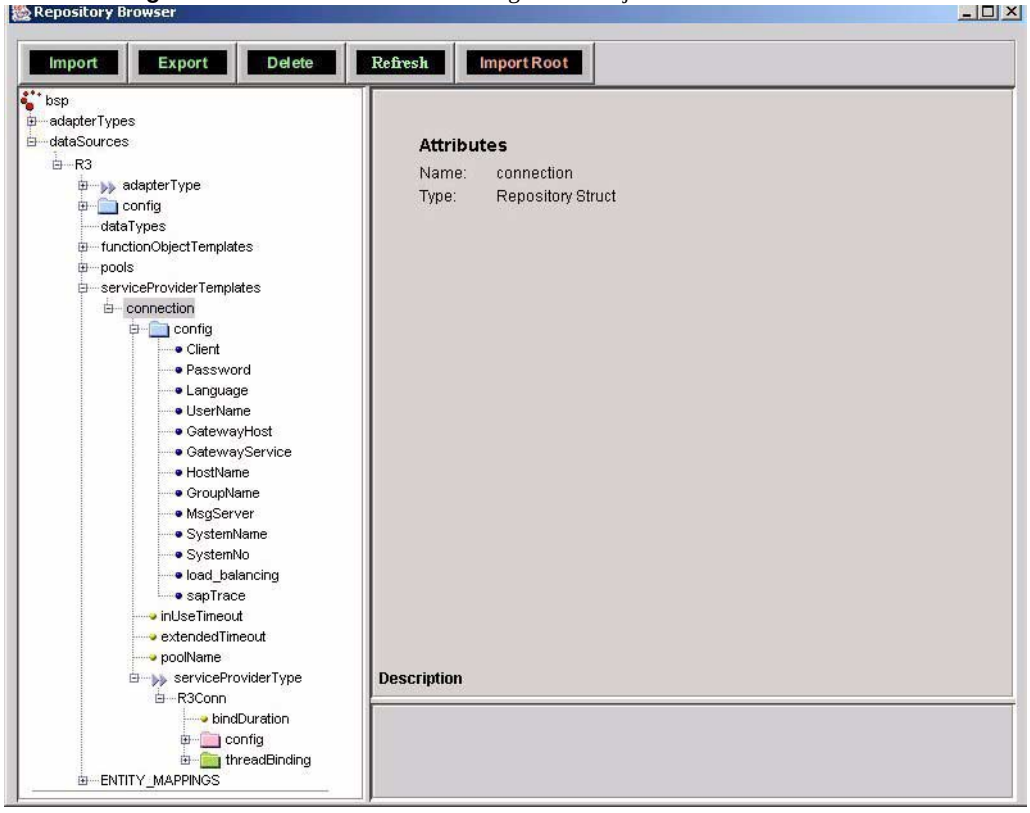


Table 2-1 lists the Service Provider fields and their definitions.

Table 2-1 Service Provider Configuration Object Field Definitions

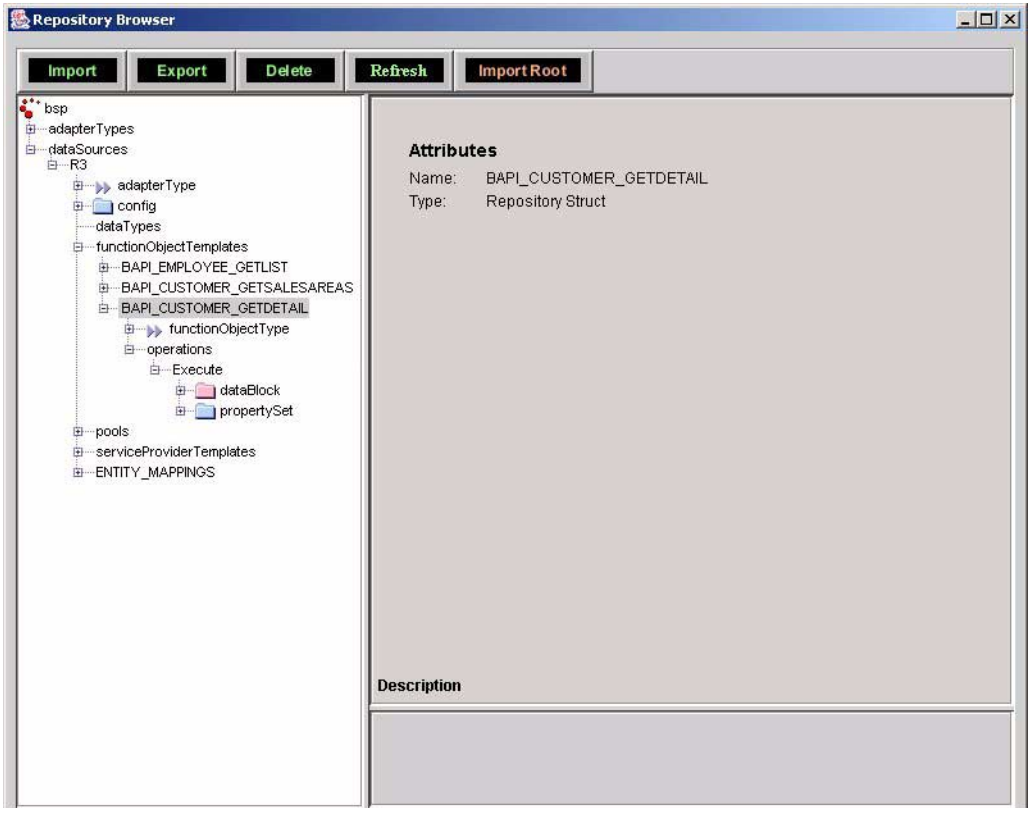
Field	Definition
Client	R/3 Client Number
Password	Password
Language	Language, for example type EN if you want English
UserName	R/3 User Name
GatewayHost	Gateway Host name
GatewayService	Gateway Service name
HostName	R/3 Application Server Host Name
Destination	Not for user use (must be null)
GroupName	Name of the specific group of application servers. For load-balancing only.
MsgServer	Host name of the message server. For load-balancing connection only.
SystemName	R/3 system name. For load-balancing connection only.
SystemNo	System number
load_balancing	Connect with: 1 - load, 0 connect without load balancing.
maxTime	Not used
sapTrace	Set R/3 trace level, 1 - trace, 0 - without trace.

Function Objects

A function object is a group of related operations that share a common state and is located under the function object template. Function object definitions represent business methods available for execution on the specific enterprise server. These are derived from metadata mined from the enterprise server.

A function object needs to be set up and associated with a service provider before it can be executed. Figure 2-3 displays the function object.

Figure 2-3 Function Object Type

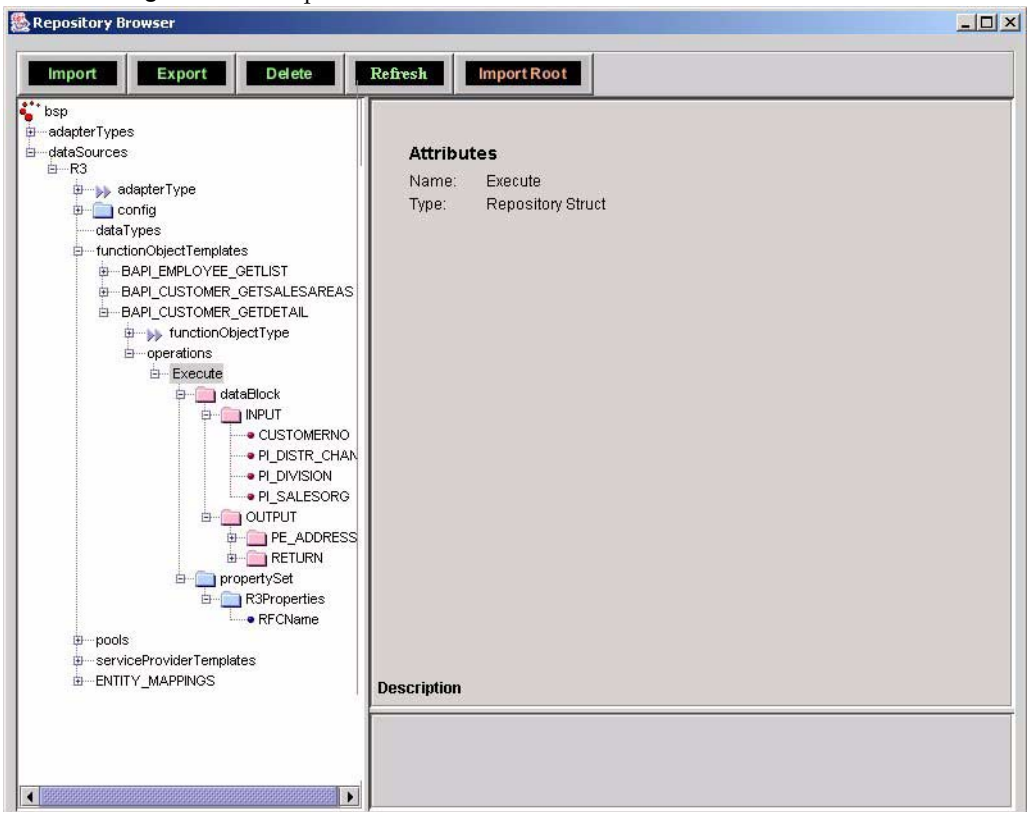


Operations

The R/3 operations node contains one type of operation: Execute.

The operations node contain data blocks and property sets as shown in Figure 2-4.

Figure 2-4 Operations



Data Block

The data block node contains two structures: input and output. The input and output structures contain fields that can be one of the following types: primitive, structure, or array.

Field Attributes

The attributes describe characteristics of fields. Figure 2-5 displays the field attributes.

The attributes are:

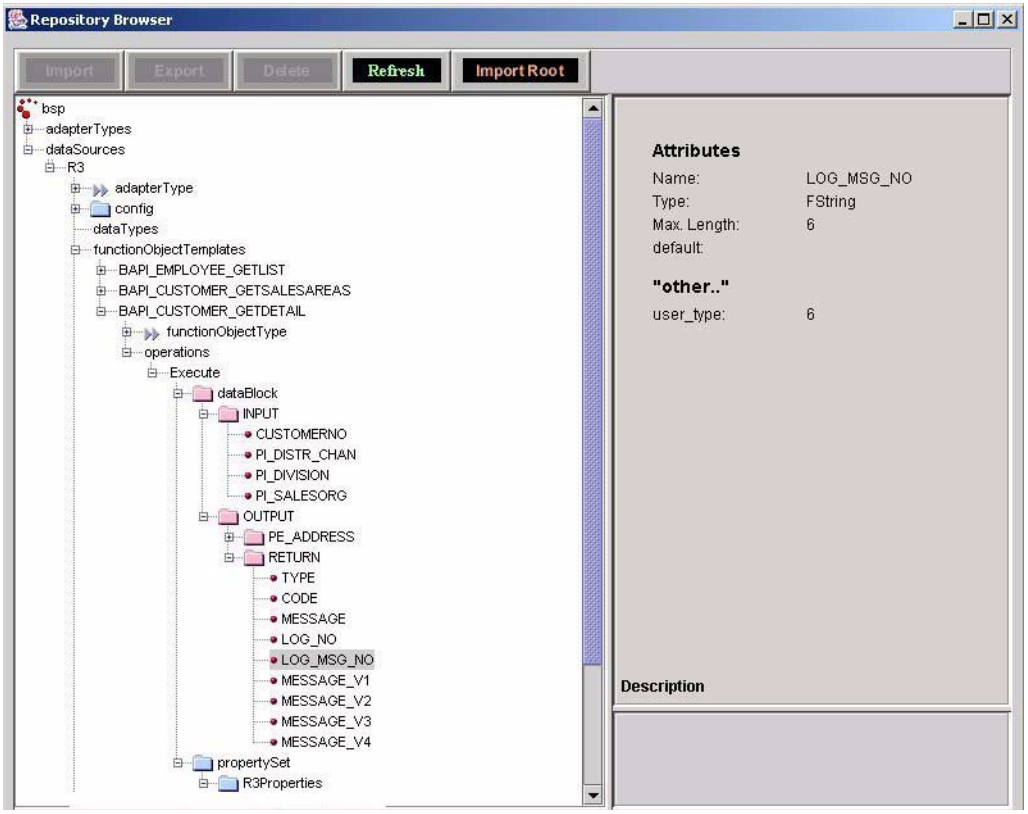
- Name - field name
- Type - field UIF type

- Max Length - maximum value length
- Default - default value that the field contains
- Others - user_type

Other Field Attributes

Every connector has its own field types in addition to the standard UIF types. Refer to Mapping R/3 Data Types to UIF Data Types for more information.

Figure 2-5 Field Attributes



Mapping R/3 Data Types to UIF Data Types

The iPlanet Application Server Enterprise Connector for R/3 uses the field attribute *user_type* to map the nonstandard UIF types.

These types are represented by the user type. In runtime the UIF type is mapped to the back-end type. Table 2-2 lists the user type, the type in R/3 and the equivalent

type in UIF.

Table 2-2 R/3 User Types

User Types	Type in R/3	Type in UIF
1	Date	FString
2	Packed	FString/Double
3	Time	FString
4	Not Used	
5	Not Used	
6	Fixed Number	FString
7	Double	Double
8	Int: 4-byte integer	Integer
9	Int2: 2-byte integer	Integer
10	Int1: 1-byte integer	Integer

Property Set

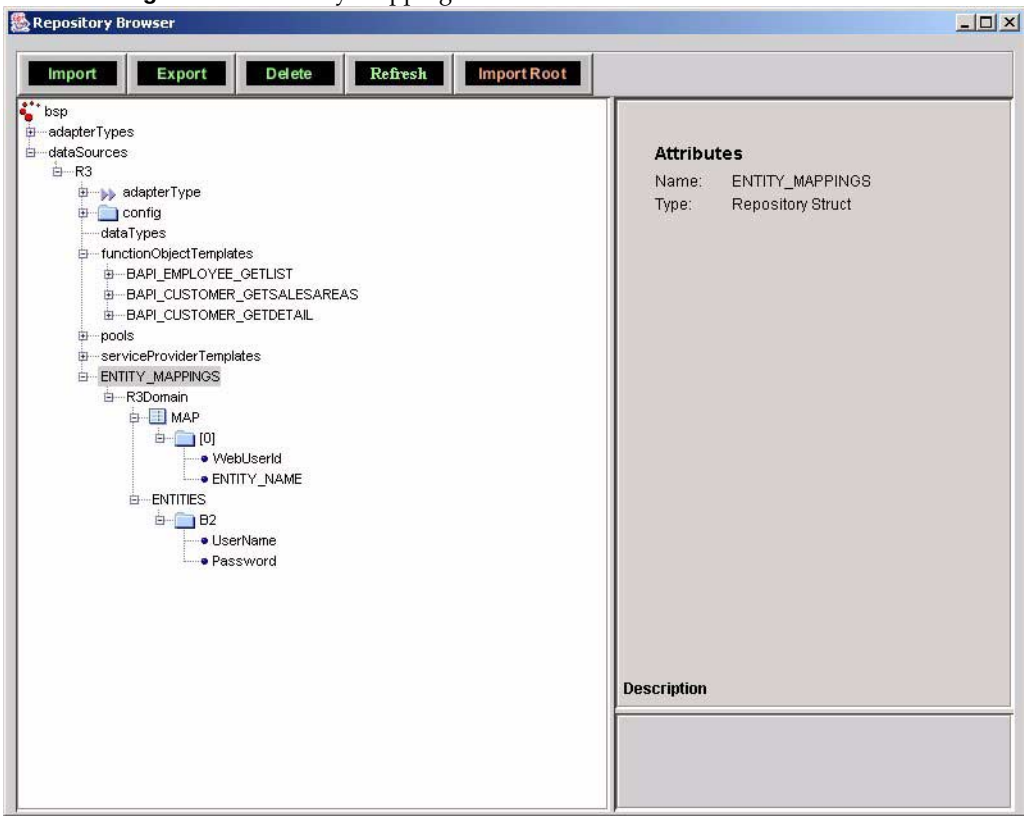
The property set contains the properties of the operation.

The R/3 property set contains one field: RFCName. It is used internally by the R/3 connector. RFCName is the name of the BAPI method that you want to activate. The value of this method matches the function object name. See Figure 2-5.

Entity Mapping

User mapping information consists of definitions of user mapping tables from the web domain to back-end domain for a specific back-end system. The contents of the user mapping tables are managed via the connector management console. See Figure 2-6 for details of the Entity Mapping. Refer to the *iPlanet Application Server Enterprise Connector for R/3 Administrator's Guide* for details on the Management Console.

Figure 2-6 Entity Mapping



The web user ID represents the web domain and is mapped to the back-end domain.

Working With Data Objects

The applications programmer needs to be able to understand and know how to use the UIF API to develop a servlet, or EJB, which communicates with the back-end system. The UIF API is an object-oriented framework.

The *iPlanet Application Server Enterprise Connector for R/3* is used to execute R/3 functions on a remote R/3 server. The servlet or EJB uses the R/3 connector to access the R/3 server.

This chapter describes the procedures for acquiring UIF objects and executing function objects.

The following topics are described:

- Data Objects
- UIF API Naming Conventions
- Working with Servlet Samples
- Deploying a Connector Application

Data Objects

A data object is used by UIF to represent data or metadata in a generic fashion.

Data objects are used to exchange data between a servlet and UIF, and between UIF and the connector.

iPlanet Application Server Enterprise Connector for R/3 allows you to access data through the data-object interface.

The data-object interface:

- presents a unified representation of back end data types

- represents complex data
- supports most common primitive data types

The types of data objects are:

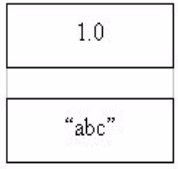
- Primitive Objects
- Structure Objects
- Array Objects
- Type Information Objects

Primitive Objects

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

- Integer, float, double
- Fixed-length string, variable-length string

Figure 3-1 Primitive Object



Integer, float, double

Integer, float, and double-data type objects hold a value whose type corresponds to the Java data type.

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.

Fixed-length string, variable-length string

Strings correspond to the Java string data type. A fixed length string has a maximum length, whereas a variable length string has no restrictions by the connector or UIF.

The maximum length of a fixed-length string is set when the string's initial value is specified, for example four characters as shown in the following line:

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

A fixed length string is truncated if it is longer than the string's initial value.

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. A structure's fields are heterogeneous.

Figure 3-2 Structure Object

"Field 1"	"Field 2"	"Field ..."
1.0	"abc"	

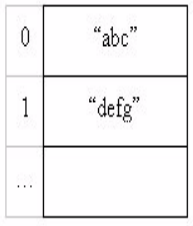
Circular references are not allowed. *iPlanet Application Server Enterprise Connector for R/3* prevents a data object being used as an attribute of itself. Indirect circular references are not checked by iPlanet Application Server for R/3.

CAUTION No error message is generated if an indirect circular reference is defined. Unpredictable results occur if a circular reference is used at runtime.

Array Objects

An array object contains data objects or primitive values as elements in the object. Array elements must be homogeneous. Each element within the array object is referred to by an integer that specifies its position in the array object.

Figure 3-3 Array Object



Type Information Objects

Type information objects are structured objects that contain the type information of a data object; for example, the definition of the fields in a structure and the field's corresponding data types. Instances of data objects can be created from type information objects. Each of these instances contain a reference to a type of information object. Numerous data types can share the same type information object.

Table 3-1 Type Information Objects

DataObjectInfo Type	Target Object
IBDPDataObject contains information about the structure of the data; for example, types, value.	IBSPDataObject
IBSPDataObjectPrimitiveInfo describes the type number, size of value (if type is string or binary), and the default value.	IBSPDataObjectPrimitive
IBSPDataObjectStructureInfo describes the type info of all fields of the target structure. The type info of each field is in turn described by a type info object.	IBSPDataObjectStructure
IBSPDataObjectListInfo describes the initial capacity and maximal element count of the target list.	IBSPDataObjectList
IBSPDataObjectArrayInfo describes the initial capacity, maximal element count and the type info of elements of the target array.	IBSPDataObjectArray

UIF API Naming Conventions

Methods in the UIF API conform to a naming convention that specifies the following:

- operation

- target
- type

The following example shows a UIF API method:

```
getElemString()
```

get is the operation, *Elem* is the target, and *String* is the type.

Operation

There are many types of operations but the two most commonly used are:

- get
- set

Target

The targets are:

- None (primitive)
- Attr (complex DataObject such as a list, array, or structure) that uses path to address attribute.

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

  The path to an element is its element number, beginning from zero. The path to a field is its field name. Element numbers and field names can be combined to create paths to attributes in complex data objects, such as a field of a structure that contains a list of elements. In this case, you specify the path as the individual attributes separated by periods (.); for example, use “field1.[01]” to identify the first element of a list at field1 in the structure.
- Elem (list/array) uses index to address element.
- Field (structure) uses name to address field.
- Current (itr) addresses object iterator is currently on.

Type

The types of operations are:

- Int
- Float
- Double
- String
- FString
- DataObject
- None

Changing Attribute Types

An attribute type can not be changed. Code Example 3-1 causes an error because it tries to change the type of a primitive object from an integer to a float.

Code Example 3-1 Changing Data Types

```
list.addElemInt (100) ; // assume 100 is added to element 1
list.setElemFloat (1, 3.14) ; // fails because the type of element is int
You can change the data type of non-primitive, as in the following example:
list.addElemDataObject (aStruct) ; // add a structure is to element 1
list.setElemDataObject (1, array) ; // change to array succeeds
```

Working with Servlet Samples

To execute an operation the servlet must be capable of the following:

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

The following examples show how to carry out 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.

Code Example 3-2 shows how to acquire a runtime object.

Code Example 3-2 Acquiring the UIF Runtime Object

```
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 the Service Provider Object

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

Code Example 3-3 shows how to create the service provider object.

Code Example 3-3 Creating the Service Provider Object

```
private IBSPServiceProvider getServiceProvider(IBSPRuntime runtime)
{
    deb.println("Before createServiceProvider()");
    if (runtime != null)
       return runtime.createServiceProvider("R3", "connection");
    else
       deb.println("runtime is null");
    return null;
}
```

Table 3-2 defines the function object parameters.

Table 3-2 Standard Provider Object Types

Parameter	Definition
R3	Data source name
connection	Service provider name

Creating Function Objects

A function object is a group of related operations that share a common state. In iPlanet Application Server for R/3, a function object needs to be set up and associated with a service provider before the function object can be executed.

Function object definitions, which represent business methods available for execution on the specific enterprise systems, are derived from metadata mined from the enterprise system.

Code Example 3-4 shows how to create the function object.

Code Example 3-4 Creating the Function Object

```
IBSPFunctionObject fn = null;
...
if(  runtime != null )
{
     deb.println("Before getServiceProvider()");
     sp = getServiceProvider(runtime);
     deb.println("After getServiceProvider()");
     if( sp != null )
     {
        deb.println("Before createFunctionObject()");
        fn = runtime.createFunctionObject(“R3”, “BAPI_EMPLOYEE_GETLIST”);
     }
}
```

Table 3-3 defines the function object parameters.

Table 3-3 Function Object Parameters

Parameter	Definition
R3	Data source name

Table 3-3 Function Object Parameters

Parameter	Definition
BAPI_EMPLOYEE_GETLIST	R3 RFC function name

Setting Up and Executing the Function Object

NOTE Commit and roll-back functions should be examined carefully when you change versions. Make sure to check R/3 documentation concerning the Commit function. See R3 release notes number 0131838, dated Feb. 11,2000, and 0192235 BAPIs.

To Set Up and Execute the Function Object

1. Specify and enable the service provider.
2. Set the web-user ID that represents the web domain, which is then mapped to the back-end domain.

   Refer to the following section, R/3 User Management, for details.
3. Prepare the function object, set up the propertySet, and set the input parameters in the function object's data block.
4. Execute the function object.
5. Retrieve the output parameters from the function block.
6. Disable the service provider.

Code Example 3-5 shows how to set up and execute the function object.

Code Example 3-5 Setting Up and Executing the Function Object

```
public void execute (String WebUserId)  throws Exception, BspException
{
       IBSPFunctionObject fn = null;
       IBSPDataObject data = null, prop = null;
    IBSPServiceProvider sp=null;
   int hr = 1;

    try
    {
```

Code Example 3-5 Setting Up and Executing the Function Object *(Continued)*

```
        IBSPRuntime runtime = getRuntime();
        if( runtime != null )
        {

            sp = getServiceProvider(runtime);
            if( sp != null )
            {
                IBSPDataObject config=sp.getConfig();
                config.setAttrFString("WebUserId",WebUserId);
                fn = runtime.createFunctionObject("R3", "BAPI_EMPLOYEE_GETLIST");
                hr = sp.enable();
                if(hr!=0)
                    return;
              if( fn != null )
                {
                    hr = fn.useServiceProvider(sp);
                    if(hr!=0)
                        return;
                    hr = fn.prepare("Execute");
                    if(hr!=0)
                        return;
                    data = fn.getDataBlock();
                    if( data != null )
                        setInputData(data);
                    prop = fn.getProperties();
                    if( prop != null )
                        setInputProperties(prop);
            hr = fn.execute();
                    if( hr == 0 )
                    {
                        data = fn.getDataBlock();
                        if(data != null )
                            getOutputData(data);
                        prop = fn.getProperties();
                        if(prop != null)
                            getOutputProperties(prop);
                    } // if(hr == 0)
                    else
                        deb.println("Execute Failed");
                } // if(fn != null)
                else
                    deb.println("Create function object Failed");
            } // if( sp != null )
            else
                deb.println("Create service provider Failed");
            hr = sp.disable();

        }
        else
            deb.println("Create runtime Failed");

    }
    catch(BspException BspError)
```

Code Example 3-5 Setting Up and Executing the Function Object *(Continued)*

```
    {
.
```

R/3 User Management

The application programmer provides a WebUserId to the R/3 Enterprise connector, which determines the R/3 authorization context to be used to process the request. The WebUserId must be set with the configuration structure of the service provider before enabling by calling the enable() method.

Code Example 3-6 illustrates how to set WebUser-test as the WebUserId before enabling the service provider.

Code Example 3-6 Setting WebUserId

```
// Create runtime
IBSPRuntime runtime = getRuntime();
if(runtime != null )
{
    // Create Service Provider
    sp = getServiceProvider(runtime);
    if(sp != null )
    {
        // Get Service Provider config structure
        IBSPDataObject config=sp.getConfig();
        // Setting WebUser-test in the WebUserId field of the config structure
        config.setAttrFString("WebUserId",WebUser-test);
        fn = runtime.createFunctionObject("R3", "BAPI_CUSTOMER_GETDETAIL");
        hr = sp.enable();
    }


}
    ........
```

Deploying a Connector Application

A developer creates an application on a development machine and then deploys the application to an application server. Deployment of an application includes installing all application files and registering all components on the destination server.

You can deploy the servlet in one of the following ways:

- Using the Deployment Tool
- Using the Command Line to Perform Deployment

In addition to deploying the servlet you must create and import the XML files, which describe the function objects, to the repository. For more details on how to do this, see the *iPlanet Application Server Enterprise Connector for R/3 Administrator's Guide*, Chapter 3 - Managing Data: The Data Mining Tool.

Using the Deployment Tool

The iPlanet Application Server Deployment Tool is a GUI-based tool that allows you to:

- Package J2EE Application Components Into Modules
- Assemble the Module Into a Deployable Unit
- Deploy the Unit to One or More iPlanet Application Server Operating Environments

Package J2EE Application Components Into Modules

J2EE application components are archived into modules according to the container that receives them upon deployment. You can archive J2EE application components into an EJB JAR module (archived with a JAR extension) or a Web Application module (archived with a WAR extension). Each module also contains a J2EE and iPlanet Application Server-specific deployment descriptors saved to XML files.

Assemble the Module Into a Deployable Unit

J2EE modules that comprise an application are assembled into a single application Enterprise Archive (EAR) file. The application EAR file also contains a J2EE deployment descriptor saved to an XML file. Depending on your requirements, the EAR file might also contain alternate deployment descriptor XML files to be used in deployment.

NOTE	EAR archives are meant to be cross-platform and will work no matter where you build the archive. For example, you can successfully deploy EAR files that you have built on Windows NT or 2000 to Solaris and vice versa.

Deploy the Unit to One or More iPlanet Application Server Operating Environments

At deployment, the EAR file is copied to the targeted iPlanet Application Server environments. Some archived application files are automatically distributed to their appropriate directories on one or more instances on the iPlanet Application Server and then registered with iPlanet Application Server; for example, static HTML files.

Using the Command Line to Perform Deployment

A Web Application Module can be deployed as a stand-alone unit or can be packaged with other modules to create an application *.ear* file. The *.ear* file contains all the modules with the application components required to run an application, along with component-level and application-level deployment descriptor files.

After you create an EAR file or a module that you may want to deploy, you may want to register it automatically via a batch file at a scheduled time and date. In this case, you would create the EAR file or module as you normally would using the Deployment Tool, but you would not use the tool for deployment.

For more detailed information on Deployment, consult the Deployment Tool Outline Help, which installs as part of the iPlanet Application Server Deployment Tool.

Chapter 4

Programming Samples

The R/3 connector is designed to facilitate development of communication between the EIS tier and the application server. Programming samples of the R/3 connector are provided to facilitate development of your own applications.

This chapter contains actual samples that show how the connector works. The following samples can be run on both Windows NT/Win2K and Solaris platforms.

- R/3 Samples
- Code Samples

R/3 Samples

The R/3 samples provided show the general flow of a connector program.

Activation

The R/3 sample consist of servlets which activate R/3 programs that access R/3 management system.

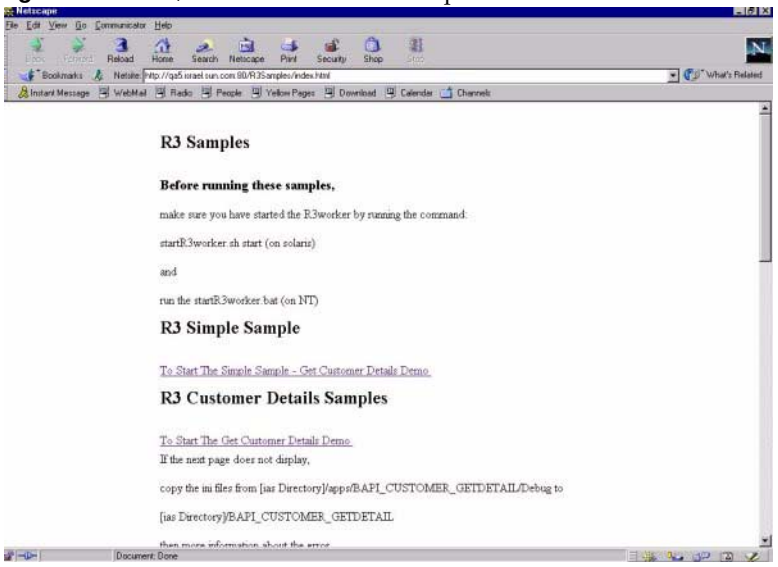
To Run the R/3 Samples on Windows NT/2000

1. Select Programs>iPlanet Application Server 6.5 >R/3 Connector 6.5 - Start Worker Process.
2. Select Programs>iPlanet Application Server 6.5>R/3 Connector 6.5 - Sample Applications.

   The R/3 sample script, displayed in Figure 4-1, has links to the samples.

3. Click on the link "To Start The Simple Sample - Get Customer Details Demo" to activate the sample shown in Figure 4-1.

Figure 4-1 R/3 Customer Details Samples



To Run the R/3 Samples on Solaris

1. Enter the following command:

   ```
   <ias directory>/ias/APPS/bin/startR3worker.sh start
   ```

2. Start your browser.

3. Enter the following URL:

   ```
   <host name>:<web server port>/R3Samples
   ```

   Run this script before running the samples.

SIMPLE_BAPI_CUSTOMER_GETDETAIL SAMPLE

The SIMPLE_ BAPI_ CUSTOMER_ GETDETAIL sample shows how to write servlets that communicate with backend systems.

To Activate the SIMPLE_BAPI_CUSTOMER_GETDETAIL Sample

1. Click on the Get Customer Details Demo hyperlink.

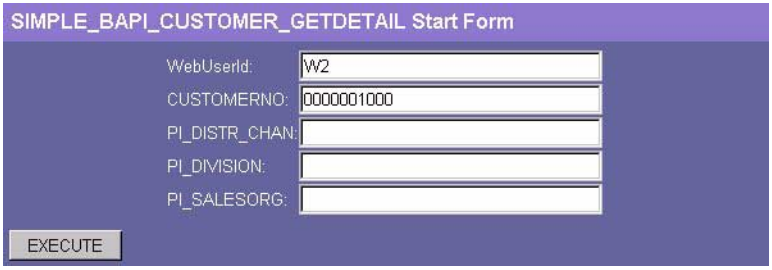
   The StartForm is displayed with the WebUserId and CUSTOMERNO, as shown in Figure 4-2.

2. Enter the following information:

   WebUserID - default is W2

   CUSTOMERNO - default is 1000

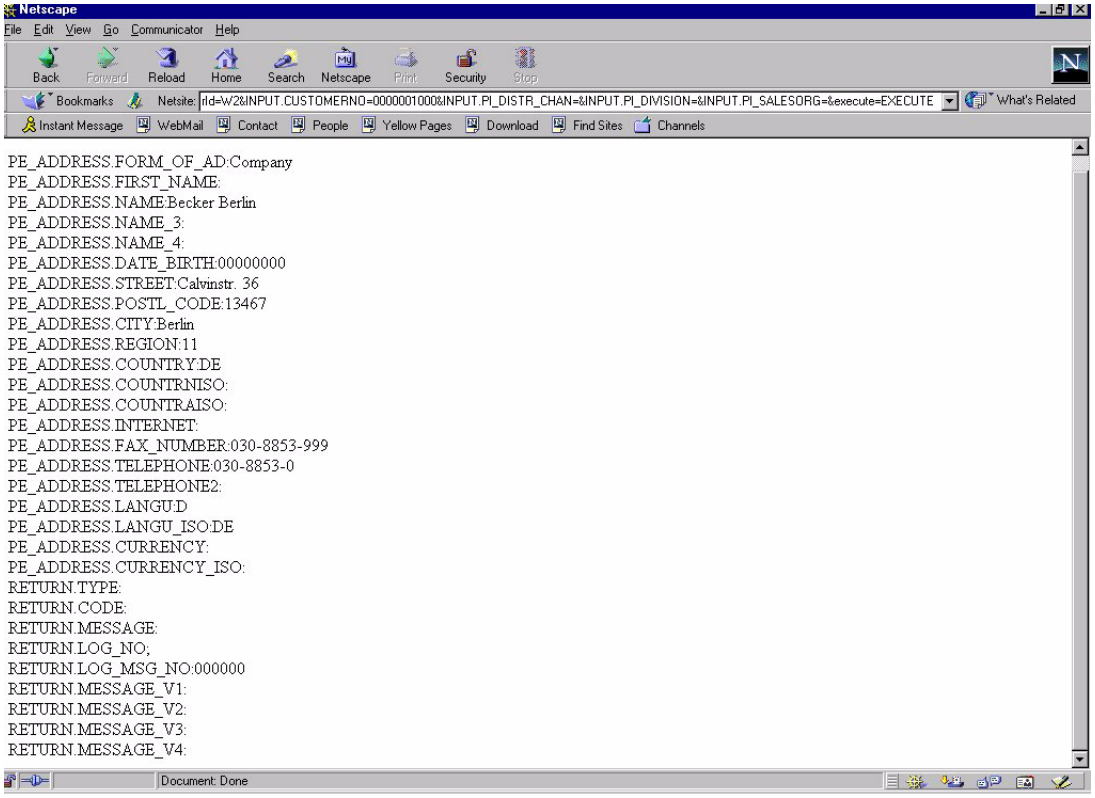
Figure 4-2 startForm.jsp



3. Click Execute.

   Figure 4-3 displays the Customer Details.

Figure 4-3 Customer Details



Code Samples

The code samples are included to show the programmer how to set input and output parameters and the BAPI call.

This is a fully operational sample. You can use it as a model for building your own application.

Code Example 4-1 and Code Example 4-2 describe the SIMPLE_BAPI_CUSTOMER_GETDETAIL sample.

Code Example 4-1 startForm.jsp

```
<HTML>
<HEAD
>  <TITLE>SIMPLE_BAPI_CUSTOMER_GETDETAIL Demo</TITLE>
</HEAD>
<BODY>
<center>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600" >
   <tr>
      <td VALIGN=TOP WIDTH="420" >
         <form action="/NASApp/SIMPLE_BAPI_CUSTOMER_GETDETAIL/main"
method=get>
         <TABLE BORDER=0 CELLSPACING=0 CELLPADDING=6 WIDTH="100%">
         <tr>
            <td BGCOLOR="#8979C8">
            <b><font face="sans-serif,arial,helvetica" color=white>
               SIMPLE_BAPI_CUSTOMER_GETDETAIL Start Form</font>
            </b>
            </td>
         </tr>
         <tr VALIGN=TOP>
            <td BGCOLOR="#666699">
            <p>
            <center>
            <TABLE BORDER=0 cellspacing=0 cellpadding=0>
            <tr>
               <td><font face="arial, helvetica, sans-serif" color=white
size="-1">WebUserId:</font></td>
               <td><font face="arial, helvetica, sans-serif">
                  <input type="text" name="WebUserId"  size=20 maxsize=50
value="W2" ></font>
               </td>
            </tr>
            <tr>
               <td><font face="arial, helvetica, sans-serif" color=white
size="-1">CUSTOMERNO:</font></td>
               <td><font face="arial, helvetica, sans-serif">
                  <input type="text" name="INPUT.CUSTOMERNO"  size=20
maxsize=50 value="0000001000" ></font>
               </td>
            </tr>
            <tr>
               <td><font face="arial, helvetica, sans-serif" color=white
size="-1">PI_DISTR_CHAN:</font></td>
               <td><font face="arial, helvetica, sans-serif">
                  <input type="text" name="INPUT.PI_DISTR_CHAN"  size=20
maxsize=50></font>
               </td>
            </tr>
            <tr>
```

Code Example 4-1 startForm.jsp

```
                <td><font face="arial, helvetica, sans-serif" color=white
size="-1">PI_DIVISION:</font></td>
                <td><font face="arial, helvetica, sans-serif">
                   <input type="text" name="INPUT.PI_DIVISION"  size=20
maxsize=50></font>
                </td>
             </tr>
             <tr>
                <td><font face="arial, helvetica, sans-serif" color=white
size="-1">PI_SALESORG:</font></td>
                <td><font face="arial, helvetica, sans-serif">
                   <input type="text" name="INPUT.PI_SALESORG"  size=20
maxsize=50></font>
                </td>
             </tr>
             </TABLE>
             </center>
             </td>
          </tr>
          </TABLE>
          <TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="100%" >
          <tr>
             <td BGCOLOR="#666699" >
             <TABLE>
             <tr>
                <td valign=top ><input type="submit" name="execute"
value="EXECUTE">
                </td>
             </tr>
             </TABLE>
             </td>
          </tr>
          </TABLE>
          </form>
       </td>
   </tr>
</TABLE>
<P>
<CENTER>
<TABLE BGCOLOR="#cccccc" BORDER="0" CELLPADDING="1" CELLSPACING="0"
WIDTH="600">
<tr>
   <td><p> </P>
      <FONT size="-2" face="PrimaSans BT, Verdana, sans-serif">Copyright ©
1999
         <A HREF="http://home.netscape.com/" NAME="Link3">Netscape
Communications Corp</A>
          All rights reserved.</FONT>
   </td>
</tr>
</TABLE>
</CENTER>
```

Code Example 4-1 startForm.jsp

```
</P>
</center>
</BODY>
</HTML>
```

Code Example 4-2 main.java

```
package SIMPLE_BAPI_CUSTOMER_GETDETAIL;

import java.util.*;
import javax.servlet.http.*;
import java.io.*;
import javax.servlet.*;
import javax.naming.*;
import netscape.bsp.*;
import netscape.bsp.runtime.*;
import netscape.bsp.dataobject.*;
import netscape.bsp.BspException.*;
public class main  extends HttpServlet
{

     protected String getInputString(HttpServletRequest request, String
parameterName,ServletOutputStream out) throws IOException
     {
        // This method is useful since getValString returns null on
        // some platforms and an empty string on other platforms for
        // missing input parameters.It also removes any whitespace
        // characters that the user my have inadvertantly entered.
        String parameter = request.getParameter(parameterName);
        if (parameter != null)
        {
            parameter = parameter.trim();
        }
        else
            out.println("\n<BR>parameter "+ parameterName+" is null");

        return parameter;
     }

     protected ServletOutputStream setOutputStream (HttpServletResponse res)
        throws ServletException, IOException
     {
        res.setContentType("text/html");
        ServletOutputStream out = res.getOutputStream();
        return out;
     }
```

Code Example 4-2 main.java

```
    private IBSPRuntime getRuntime()   throws BspException
    {
       com.kivasoft.IContext _ctx =
((com.netscape.server.servlet.platformhttp.PlatformServletContext)
getServletContext()).getContext();
       IBSPRuntime ibspruntime = access_cBSPRuntime.getcBSPRuntime(_ctx, null,
null);
       return ibspruntime;
    }

    private IBSPServiceProvider getServiceProvider(IBSPRuntime runtime)
throws BspException
    {
       if(runtime != null)
          return runtime.createServiceProvider("R3", "connection");
       else
          return null;
    }

    public void doGet (
       HttpServletRequestrequest,
       HttpServletResponseresponse ) throws ServletException, IOException
    {
       int hr=0;
       IBSPServiceProvider sp=null;
       IBSPFunctionObject fn = null;
       IBSPDataObject data = null, prop = null;
       ServletOutputStream out = setOutputStream(response);
       try
       {
          String WebUserId=new
String(getInputString(request,"WebUserId",out));
          IBSPRuntime runtime = getRuntime();
          if(runtime != null )
          {
             sp = getServiceProvider(runtime);
             if(sp != null )
             {
                IBSPDataObject config=sp.getConfig();
                config.setAttrFString("WebUserId",WebUserId);
                fn = runtime.createFunctionObject("R3",
"BAPI_CUSTOMER_GETDETAIL");
                hr = sp.enable();
                if(hr!=0)
                   return;
                if( fn != null )
                {
                   hr = fn.useServiceProvider(sp);
                   if(hr!=0)
                      return;
                   hr = fn.prepare("Execute");
                   if(hr!=0)
```

Code Example 4-2 main.java

```
                        return;
                    data = fn.getDataBlock();
                    setInputData(request,data,out);
                    hr = fn.execute();
                    if( hr == 0 )
                    {
                        data = fn.getDataBlock();
                        if(data != null )
                            getOutputData(out,data);
                    } // if(hr == 0)
                    else
                        out.println("\n<BR>Execute Failed");
                } // if(fn != null)
                else
                    out.println("\n<BR>Create function object Failed");
            } // if( sp != null )
            else
                out.println("\n<BR>Create service provider Failed");
            hr = sp.disable();
        }
        else
            out.println("\n<BR>Create runtime Failed");
      }//try.
      catch(BspException BspError)
      {
          if(sp!=null)
          {
              hr = sp.disable();
          }
          String errorMessage="BspException:"+BspError.getMessage();
          IBSPDataObjectStructure info = null;
          info=BspError.getInfo();
          if (info != null && info.attrExists("msgid") )
          {
              errorMessage +=  " Error code : " + info.getAttrInt("msgid") ;
          }
          out.println("\n<BR>"+errorMessage);
      }
      catch(Exception exception)
      {
          if(sp!=null)
          {
              hr = sp.disable();
          }
          String errorMessage="Exception:"+exception.toString();
          out.println("\n<BR>"+errorMessage);
      }
      return;
    }//doGet.

    public void setInputData(HttpServletRequest request,IBSPDataObject
data,ServletOutputStream out)    throws BspException,IOException
    {
```

Code Example 4-2 main.java

```
data.setAttrFString("INPUT.CUSTOMERNO",getInputString(request,"INPUT.CUSTOMERNO
",out));

data.setAttrFString("INPUT.PI_DISTR_CHAN",getInputString(request,"INPUT.PI_DIST
R_CHAN",out));

data.setAttrFString("INPUT.PI_DIVISION",getInputString(request,"INPUT.PI_DIVISI
ON",out));

data.setAttrFString("INPUT.PI_SALESORG",getInputString(request,"INPUT.PI_SALESO
RG",out));
     }

     public void getOutputData(ServletOutputStream out,IBSPDataObject data)
throws BspException,IOException
     {

out.println("\n<BR>PE_ADDRESS.FORM_OF_AD:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.FORM_OF_AD"));

out.println("\n<BR>PE_ADDRESS.FIRST_NAME:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.FIRST_NAME"));

out.println("\n<BR>PE_ADDRESS.NAME:"+data.getAttrFString("OUTPUT.PE_ADDRESS.NAM
E"));

out.println("\n<BR>PE_ADDRESS.NAME_3:"+data.getAttrFString("OUTPUT.PE_ADDRESS.N
AME_3"));

out.println("\n<BR>PE_ADDRESS.NAME_4:"+data.getAttrFString("OUTPUT.PE_ADDRESS.N
AME_4"));

out.println("\n<BR>PE_ADDRESS.DATE_BIRTH:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.DATE_BIRTH"));

out.println("\n<BR>PE_ADDRESS.STREET:"+data.getAttrFString("OUTPUT.PE_ADDRESS.S
TREET"));

out.println("\n<BR>PE_ADDRESS.POSTL_CODE:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.POSTL_CODE"));

out.println("\n<BR>PE_ADDRESS.CITY:"+data.getAttrFString("OUTPUT.PE_ADDRESS.CIT
Y"));

out.println("\n<BR>PE_ADDRESS.REGION:"+data.getAttrFString("OUTPUT.PE_ADDRESS.R
EGION"));
```

Code Example 4-2 main.java

```
out.println("\n<BR>PE_ADDRESS.COUNTRY:"+data.getAttrFString("OUTPUT.PE_ADDRESS.
COUNTRY"));

out.println("\n<BR>PE_ADDRESS.COUNTRNISO:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.COUNTRNISO"));

out.println("\n<BR>PE_ADDRESS.COUNTRAISO:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.COUNTRAISO"));

out.println("\n<BR>PE_ADDRESS.INTERNET:"+data.getAttrFString("OUTPUT.PE_ADDRESS
.INTERNET"));

out.println("\n<BR>PE_ADDRESS.FAX_NUMBER:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.FAX_NUMBER"));

out.println("\n<BR>PE_ADDRESS.TELEPHONE:"+data.getAttrFString("OUTPUT.PE_ADDRES
S.TELEPHONE"));

out.println("\n<BR>PE_ADDRESS.TELEPHONE2:"+data.getAttrFString("OUTPUT.PE_ADDRE
SS.TELEPHONE2"));

out.println("\n<BR>PE_ADDRESS.LANGU:"+data.getAttrFString("OUTPUT.PE_ADDRESS.LA
NGU"));

out.println("\n<BR>PE_ADDRESS.LANGU_ISO:"+data.getAttrFString("OUTPUT.PE_ADDRES
S.LANGU_ISO"));

out.println("\n<BR>PE_ADDRESS.CURRENCY:"+data.getAttrFString("OUTPUT.PE_ADDRESS
.CURRENCY"));

out.println("\n<BR>PE_ADDRESS.CURRENCY_ISO:"+data.getAttrFString("OUTPUT.PE_ADD
RESS.CURRENCY_ISO"));

out.println("\n<BR>RETURN.TYPE:"+data.getAttrFString("OUTPUT.RETURN.TYPE"));

out.println("\n<BR>RETURN.CODE:"+data.getAttrFString("OUTPUT.RETURN.CODE"));

out.println("\n<BR>RETURN.MESSAGE:"+data.getAttrFString("OUTPUT.RETURN.MESSAGE"
));

out.println("\n<BR>RETURN.LOG_NO;"+data.getAttrFString("OUTPUT.RETURN.LOG_NO"))
;

out.println("\n<BR>RETURN.LOG_MSG_NO:"+data.getAttrFString("OUTPUT.RETURN.LOG_M
SG_NO"));
```

Code Example 4-2 main.java

```
out.println("\n<BR>RETURN.MESSAGE_V1:"+data.getAttrFString("OUTPUT.RETURN.MESSA
GE_V1"));

out.println("\n<BR>RETURN.MESSAGE_V2:"+data.getAttrFString("OUTPUT.RETURN.MESSA
GE_V2"));

out.println("\n<BR>RETURN.MESSAGE_V3:"+data.getAttrFString("OUTPUT.RETURN.MESSA
GE_V3"));

out.println("\n<BR>RETURN.MESSAGE_V4:"+data.getAttrFString("OUTPUT.RETURN.MESSA
GE_V4"));
    }
}//main
```

Appendix A

Error Messages

The developer writes servlets that use connectors to connect with the back end. Error messages are generated to assist the developer in debugging these processes. This appendix contains a list of identified errors and the error handling code that is used to identify the error.

The following topics are described:

- Description of Errors
- Error Handling Code

Description of Errors

Table A-1 lists the error handling codes, *msgid*, and the corresponding messages.

Table A-1 Error Messages

Error Message Code ***msgid***	**Error Message**
1	Communication Failure. Error: {error text} (rc : {return code}), Applind : {indication}.
2	Failed to connect R/3, RFC Error: Key:{R/3 return code}, Status:{R/3 return status}, ErrorMessage:{error text}.
3	Output data is too long, maximal length {length}, actual length {length}.
4	Illegal user type value {type code} for property name {name}.
5	Internal error. Failed to get object {object type} {object name}.
6	Internal error. Object {object name} is NULL.

Table A-1 Error Messages

Error Message Code *msgid*	Error Message
7	Type not supported, in Data Object {object name}, field {field name}, type {field type}.
8	Not used.
9	Not used.
10	Not used.
11	Not used.
12	Not used.
13	Not used.
14	Not used.
15	Not used.
16	R/3 memory exception.
17	R/3 Failed to create function object {function name}, Key:{R/3 return code}, Status:{R/3 return status}, ErrorMessage:{error text}.
18	R/3 function call Error, function name : { function name }, Return code:{numeric code}. ErrorMessage: { error text }.
19	R/3 function call Error, function name : { function name }, Key:{R/3 return code}, Status:{R/3 return status}, ErrorMessage:{error text}.

Error Handling Code

The error handling code is an example of how to handle an exception raised by the connector.

When there is an error in the process, the *iPlanet Application Server Enterprise Connector for R/3* throws a BspException object. The exception object contains a data object. The data object contains the numeric code, *msgid*, as listed in Table A-1.

An example of error handling code is shown in Code Example A-1.

Code Example A-1 Error Handling Code Sample

```
try
{
   ...  some code in the servlet
}
        catch (BspException e)
{
                IBSPDataObjectStructure info = null;
                errorMessage += " Error : " + e.getMessage() ;
                info=e.getInfo();
                if (info != null && info.attrExists("msgid") )
                {
                  errorMessage +=  " Error code : " + info.getAttrInt("msgid")
;
                }
}
```

Glossary

API (Application Programming Interface) Software that an application utilizes to carry out and request lower-level services by the operating system. In addition, a set of standard software data formats that application programs use to initiate contacts with other programs, computers, and systems.

Applet A Java program that can be distributed as an attachment in a World Wide Web document and executed in a Java-enabled web browser.

Applications Programmer Responsible for writing servlets or EJBs that call the UIF API. Uses the Repository Browser to determine the available data types and access methods.

Array Object Contains data objects or primitive values as elements in the object. Array elements must be homogeneous. Each element within the array object is referred to by an integer that specifies its position in the array object.

Attribute Field Attributes that describe allowable attributes for the field where the input and output are located.

CICS (Customer Information Control System) An IBM communications program designed to allow transactions entered at a remote site to be processed concurrently by a mainframe host.

Connection Parameters Contains information needed to connect to the R/3 system.

Daemon A program that is not explicitly invoked, and remains idle until summoned (called on).

Data Block Describes the input and output of operations. The data block can only contain two structures: input and output. All input and output structures contain fields that can be only one of the following types: primitive, structure, or array.

Data Object Used by the UIF to represent data or metadata in a generic fashion. Data objects are used to exchange data between a servlet and the UIF, and between the UIF and the connector.

Data Source Contains all the information needed to connect to the R/3 system, and stores all the function objects. In addition, the Data Source determines which system to mine, and where to place the function objects.

Deployment Deploying an application includes installing all of the application's files, and registering all of its components on the destination server. You deploy an application using the Deployment Tool, a separate tool accessible from the iPlanet Application Server (iAS). An application must be deployed before it can be used.

EJB (Enterprise Java Beans) A server-side component architecture for writing reusable business logic and portable enterprise applications. They are written entirely in Java and run on any EJB compliant server. They are operating system, platform, and middleware independent, thereby preventing vendor lock-in.

EIS (Enterprise Information System) Referred to as a backend system.

Enterprise Connector The component in iPlanet Application Server Enterprise Connector for R/3, PeopleSoft, Tuxedo, or CICS that enables you to access the appropriate backend system.

ERP (Enterprise Resource Planning) A multi-module software system that supports enterprise resource planning. An ERP system typically includes a relational database and applications for managing purchasing, inventory, personnel, customer service, shipping, financial planning, and other important aspects of the business.

Function Object A group of business methods available for execution on the specific enterprise server. These objects are derived from metadata mined from the enterprise server that share a common state.

Group Name Name of the specific group of application servers. For load-balancing only.

iPlanet Application Server The iAS provides the most robust e-commerce platform for delivering innovative and leading-edge application services to a broad range of servers, clients, and devices.

iWS (iPlanet Web Server) A web server that is ideally suited to the Java development community for use as the development and test platform for web applications.

Java An object-oriented programming language developed by Sun Microsystems, Inc. to create executable content (i.e, self-running applications) that can be easily distributed through networks like the Internet.

Load Balancing Load Balancing is the configuration of a computer system, network, or disk subsystem to more evenly distribute the data and/or processing across available resources in order to increase the speed and reliability of transmissions.

MsgServer Host name of the message server. For load-balancing connection only.

Operations Directory A directory with operations that contain data blocks and property sets.

Primitive Object A data type that contains a single value of an integer, float, double, fixed-length string, or variable-length string.

Repository A specialized structure where all the module's functions are stored for the use of the iPlanet Application Server Enterprise Connector.

Repository Browser The component that enables you to browse data (content) in the repository, and to view the available functions (input and output parameters) for the backend system.

Runtime Object The entry point into the UIF.

Service Provider Object The logical representation of a connection to a back-end system, which must be enabled before it can be used. Typically, the service provider object is not bound to a physical connection until absolutely necessary.

Server Tier The server tier is represented by an application server and optionally a web server such as the iPlanet Web Server Enterprise Edition. The server tier houses the business logic (Enterprise Java Beans of your application servlets), and provides scalability, high availability load balancing, and integration with a variety of data sources.

Servlet An applet that runs on a server, usually meaning a Java applet that runs on a Web server.

Structure Object Contains other data objects or primitive values whose fields are heterogeneous such as as fields, and whose fields are heterogeneous. Each object within the structure object is referred to by a string that represents the field name. Field names have a maximum length of 32 characters.

System Name The system name used. For load-balancing connection only.

Three-tier Application Model A model of an application system that is composed of the following three tiers: Client, Server, and Backend (EIS).

Type Information Objects Structured objects that contain the type information of a data object; i.e. definition of the fields in a structure and the fields corresponding data types. Instances of data objects can be created of type information objects. Each of these instances contain a reference to a type of information object. Numerous data types can share the same type information object.

UIF (Unified Integration Framework) An application programming framework that provides a single Application Programming Interface (API) to access different backend systems.

URL (Universal Resource Locator) An address for a resource or site (usually a directory or file) on the World Wide Web, and the convention that web browsers use for locating files and other remote services.

XML (eXtensible Markup Language) A common cross-platform format document used to populate a repository.

Worker A worker is an out-of-process unthreaded procedure. The conversation to the backend system is done by the worker process. The worker returns the results to the connector using the proprietary protocol.

Index

M

O

R

S

T

U

W