Developer's Guide

iPlanet Application Server Enterprise Connector for CICS

Version 6.5

806-5504-02
August 2002

Contents

List of Figures

List of Tables

List of Procedures

List of Code Examples

Preface

The *iPlanet Application Server Enterprise Connector for CICS Developer's Guide* gives a brief overview of the CICS connector and explains how to write the servlet or Enterprise Java Bean (EJB) applications. The *iPlanet Application Server Enterprise Connector for CICS Developer's Guide* is written for application programmers who develop internet or intranet applications for the CICS Enterprise Information System (EIS).

This preface contains information about the following topics:

- Prerequisites
- What's in This Guide
- Documentation Conventions
- Guide Online
- 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
- CICS 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 CICS Developer's Guide* covers the information you need to know to write servlets or EJBs that utilize UIF and the iPlanet Application Server Enterprise Connector for CICS to connect to your CICS EIS.

The following table summarizes the purpose of each chapter..

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"	Acquire UIF 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 EIS.
Chapter 4, "Programming Examples"	View a simple sample application and view examples of code that can be used to set up the CICS Enterprise Connector.
Appendix A, "Error Messages and Codes"	Display error messages for CICS Enterprise Connector and their codes. This appendix also includes a sample of the error handling code that can be used by the servlet developer.
Appendix B, "Communication Failure Codes"	Display a list of the most common causes for communication failure, and the recommended action to be taken.

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.

Guide Online

You can find the *iPlanet Application Server Enterprise Connector for CICS 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 for administrators, end users and developers. Use the following URL to view the related documentation:

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

The following lists the documents that are available:

- *iPlanet Application Server Enterprise Connector for CICS 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 Unified Integration Framework Release Notes*

- *iPlanet Unified Integration Framework Developer's Guide*

Chapter 1

Overview

The iPlanet Application Server Enterprise Connector for CICS is used for building and delivering scalable applications that integrate the application server with legacy CICS applications. The iPlanet Application Server Enterprise Connector for CICS enables communication between an end user and a remote CICS Enterprise Information System (EIS). This chapter introduces the three-tier, web-based computing model and describes the basic connector concepts.

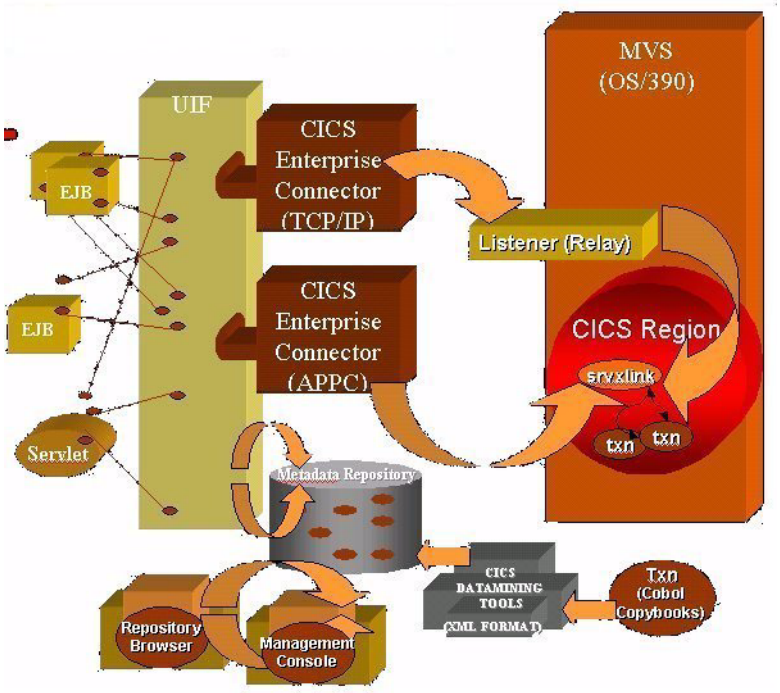
This chapter contains information about the following topics:

- Unified Integration Framework
- The Three-tier Application Model
- Architecture of the CICS Connector
- Enterprise Connector Tools for CICS

Unified Integration Framework

The Unified Integration Framework (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 EIS to allow communication between the UIF API and the EIS, see Figure 1-1. The UIF API is the only API necessary to access the EIS.

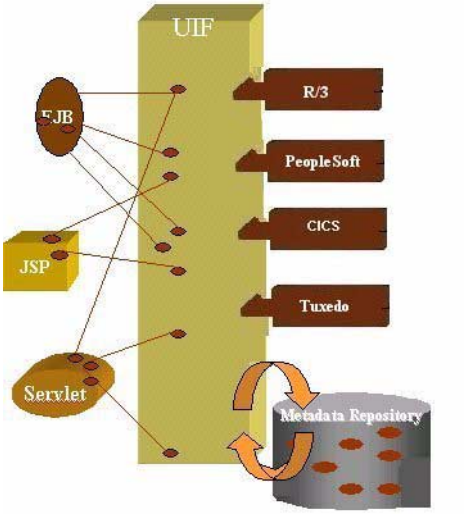
Figure 1-1 CICS Integration



The UIF enables development of server extensions that integrate with legacy CICS applications and systems, client-server applications, and third-party Internet solutions. These extensions provide a consistent access layer to disparate EIS, dramatically reducing development effort. The framework provides support for features such as object pooling, 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 EIS. For example, the metadata often describes the physical connection between systems, the data that is available, and methods you can use to process data. See Figure 1-2.

Figure 1-2 The Unified Integration Framework



Since EISs are implemented differently, the details for each EIS implementation differ.

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 "The Three-tier Application Model" for a description. The UIF mediates between the iPlanet Application Server application and the EIS tier, namely the data sources and databases.

The UIF provides an API to access the following services:

- Runtime
- Data Object Services
- 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 implements universal data representation common to all connectors. See Chapter 3, “Working With Data Objects” for 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. Italso manage the instances and reuse of data objects from datatype definitions.

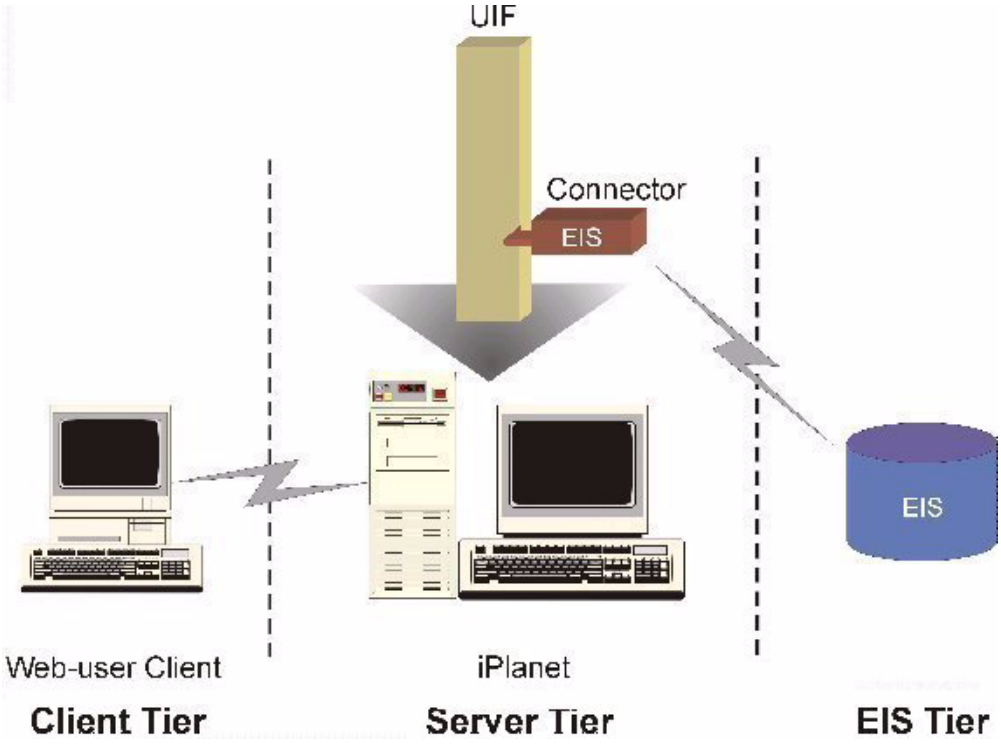
The Three-tier Application Model

The machine and software involved are divided into the following three tiers:

- Client Tier
- Server Tier
- EIS Tier

The connectors serve as an essential link allowing the server tier to communicate with the EIS tier, as shown in Figure 1-3. Communication between the application server and the 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 EISs in a seamless and uniform manner.

Figure 1-3 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 a Java applet).

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 Tier

The EIS tier is represented by Enterprise Resource Planning (ERP) systems or other EIS data systems such as CICS.

Architecture of the CICS Connector

The iPlanet Application Server Enterprise Connector for CICS allows you to build interactive web-based e-Business applications. Specifically, the iPlanet Application Server Enterprise Connector for CICS allows you to retrieve, display, and manipulate data within a CICS program that is initiated by an EJB or servlet.

The iPlanet Application Server Enterprise Connector for CICS stores data that defines the services available to a servlet or EJB from a CICS system in the repository. Repository data includes metadata definitions of Data Sources and Service Provider Types. These define general characteristics of CICS connector and configuration information about particular CICS systems.

A set of enterprise tools provides graphic user interfaces to help you manipulate the enterprise connector enabled CICS functionality.

Mainframe components

Installation of iPlanet Application Server for CICS on the mainframe includes the following modules:

- SRVXLINK, which automatically transfers data between iPlanet Application Server for CICS and a CICS program, passing data via the COMMAREA.
- Relay (TCP/IP only).

If your CICS program follows Distributed Program Link (DPL) guidelines, you do not need to modify your CICS program; you can use the SRVXLINK module, which automatically transfers data between CICS program (via the COMMAREA) and the iPlanet Application Server.

If your program does not follow DPL guidelines, you must modify the program to follow the DPL guidelines.

Connector-to-CICS Communication

The iPlanet Application Server Enterprise Connector for CICS communicates with the mainframe via TCP/IP or APPC.

The connector does not directly invoke user CICS transactions, rather the connector invokes the SRVX transaction (default) of the SRVXLINK program. The connector passes the user transaction name and any input parameters to SRVXLINK. SRVXLINK then invokes the user transaction, passing the input data via the COMMAREA.

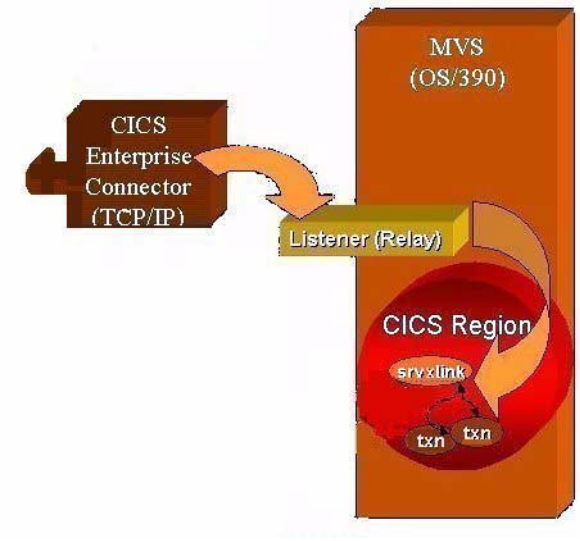
The connection with the CICS back end is either persistent or non-persistent. Persistent connections are achieved by specifying a REQUEST field in the repository as SendLast. This causes a re-use of connections in the pool, and enable scaling to high transaction rates. This is the recommended mode of operation.

Non persistent connections are achieved by specifying a REQUEST field in the repository as SendAndEnd. This opens a new connection for each request. This mode of operation is suitable for infrequent (occasional) requests to CICS EIS.

Connector-to-CICS Architecture Over TCP/IP

iPlanet Application Server for CICS enabled servers can communicate with CICS regions. Figure 1-4 shows the interaction between the server and the MVS machine that hosts CICS over TCP/IP.

Figure 1-4 Connector-to-CICS Architecture Over TCP/IP



A Listener/Relay component resides on the OS/390 (MVS) operating system. It connects the iPlanet Application Server Enterprise Connector for CICS server, which communicates via TCP/IP, to the CICS system, which uses the SNA/APPC protocol (LU6.2).

The TCP/IP conversation with the application server is persistent, whereas the conversations with the CICS transactions can be nonpersistent. To achieve scalability, the connections to the relay are pooled in iPlanet Application Server Enterprise Connector for CICS and can be re-used by many other users. Each connection can be used to conduct many transactions with the CICS system. For example, when a user issues a request to run a new transaction in CICS, a connection is pulled from the pool and the request for a new transaction is forwarded to the relay. The relay then allocates a new conversation with CICS, which invokes a new transaction. For more information about pooling see the *iPlanet Application Server Enterprise Connector for CICS Administrator's Guide.*

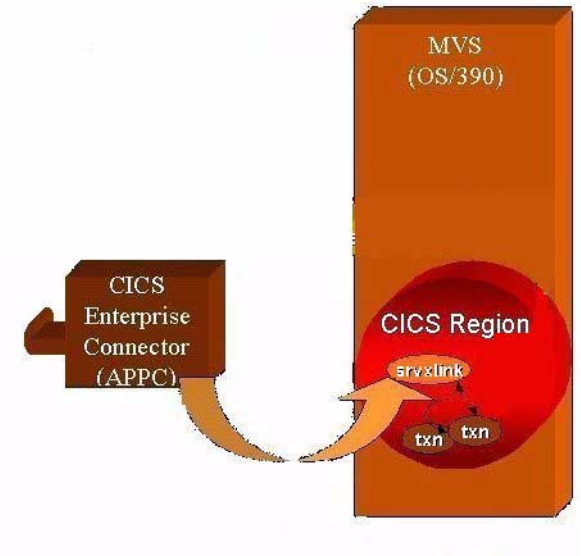
Connector-to-CICS Architecture Over APPC

A SRVX transaction resides on OS/390 (MVS). It connects the iPlanet Application Server for CICS application server to the CICS server, which communicates via APPC to the CICS system.

One or more iPlanet Application Server for CICS enabled servers can communicate with CICS regions.

Figure 1-5 shows the interaction between the server and the MVS machine that hosts CICS over APPC.

Figure 1-5 Connector-to-CICS Architecture Over APPC:



Enterprise Connector Tools for CICS

The Enterprise Connector Tools are as follows:

- Management Console - includes the User Mapping and Data Mining Tools
  - User Mapping - allows you to map user ID's for access into the EIS.
  - Data Mining Tool - includes capabilities of determining the available functions in the EIS, 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 EIS. For developer's use of the Repository Browser see Chapter 2.

For more information about these tools, refer to the *iPlanet Application Server Enterprise Connector for CICS Administrator's Guide*.

Chapter 2

Viewing the Repository Contents

The Repository Browser is designed to provide the developer with a convenient tool 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 developer must be able to see the contents of the repository to be able to program an application. Variable values in the repository can not be changed using the repository browser. XML files may be imported and exported using the repository browser but use of this feature is not recommended.

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 Access the Repository Browser from Windows NT/2000

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

To Access the Repository Browser from Solaris

1. Enter the following command lines:

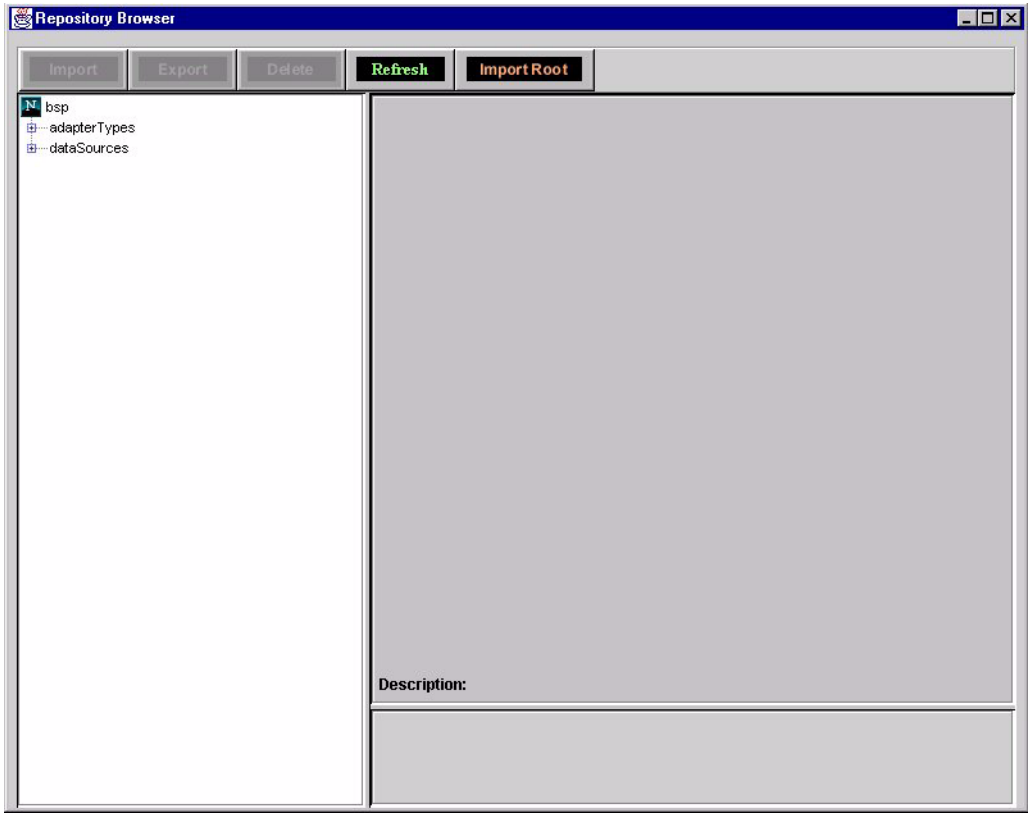
   ```
   cd iplanet/ias6/ias/APPS/bin
   ```

   and

   ```
   ./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 data sources. These nodes are hierarchical and can be expanded to show details of the data structure and function objects. The right pane displays the properties and values of the node selected in the left pane.

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 an EIS. Usually, the service provider object is not bound to a physical connection until it is absolutely necessary. The service provider object is under the service provider template as shown in Figure 2-2.

Figure 2-2 Service Provider Configuration Object

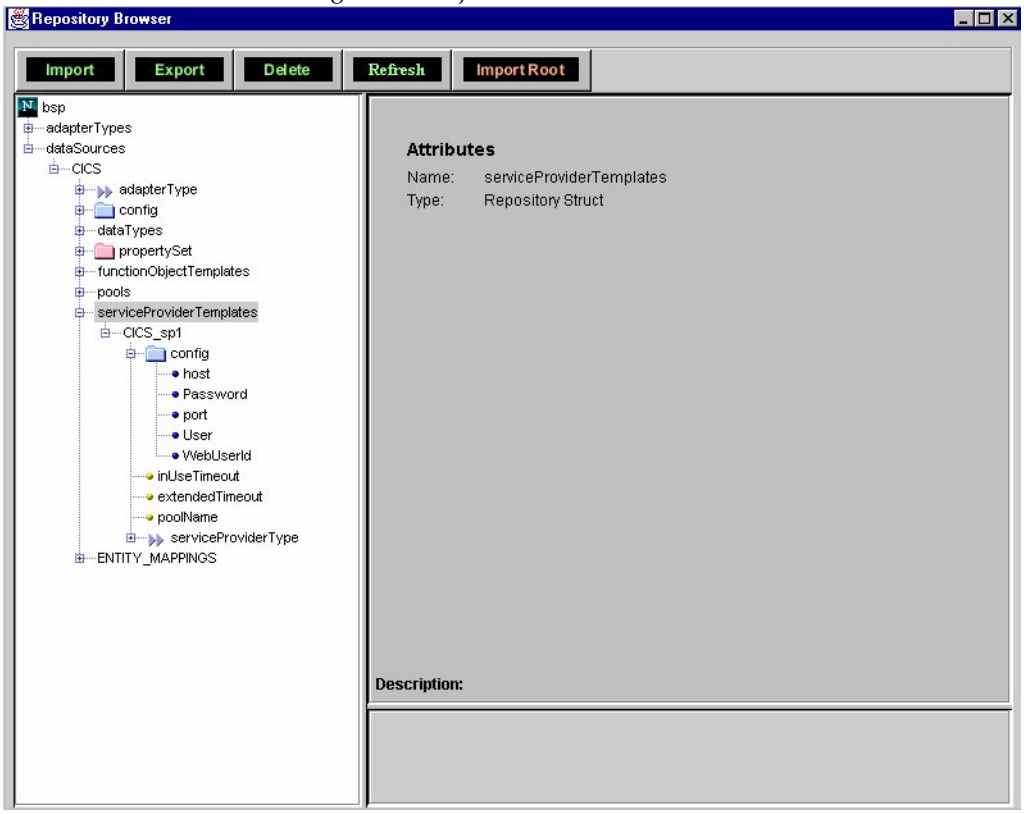


Table 2-1 lists the fields and the definitions of all the fields.

Table 2-1 Service Provider Configuration Object Field Definitions

Field	Definition
host	Host machine name of the target machine.
Password	CICS user password.
port	TCP/IP port that is used to connect to the Relay.
User	CICS user name.
WebUserId	ID for user mapping.

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 shows the function object.

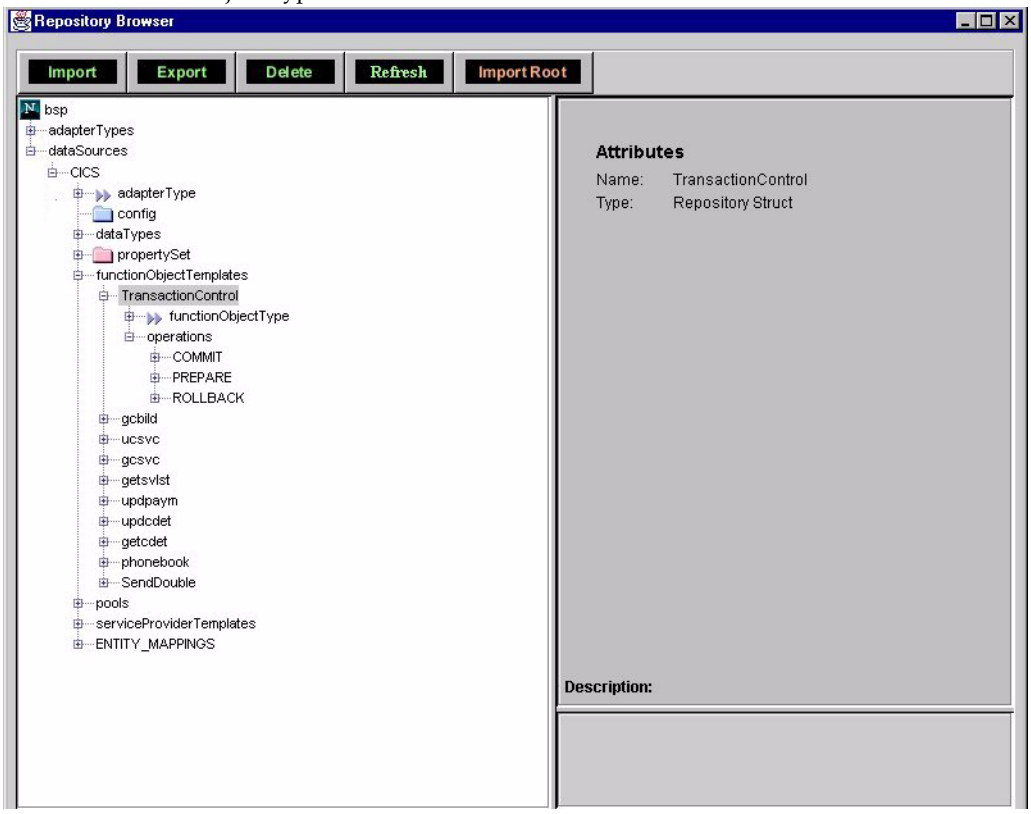
Function Object Template

The function object template includes function objects. The function object TransactionControl contains the following operations:

- COMMIT — Used to commit the transaction.
- PREPARE — Used to prepare the transaction for a commit.
- ROLLBACK — Used to rollback the changes that were done in this transaction.

Other function objects represent CICS programs. Each of these nodes contain operations.

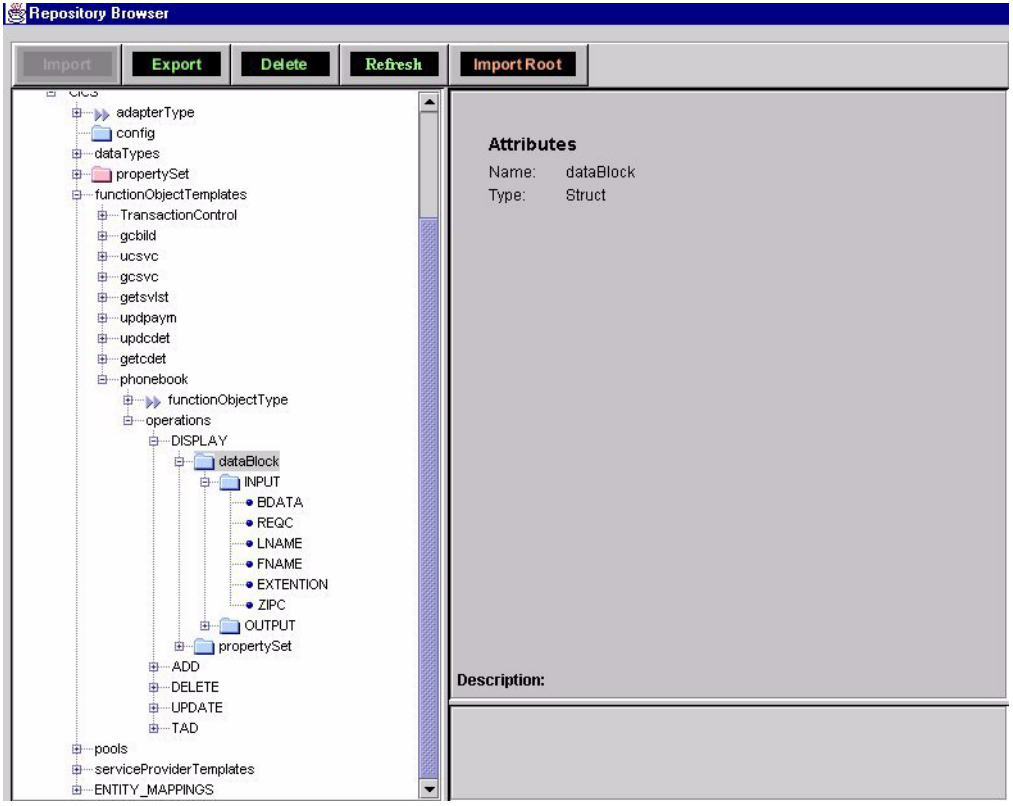
Figure 2-3 Function Object Type



Operations

The operation node contains the dataBlocks and propertySet nodes as shown in Figure 2-4.

Figure 2-4 Operations



dataBlock

The dataBlock 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 connected to the following fields:

- Name - field name

  Type - field UIF type

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

Other Field Attributes

Every connector has its own field types in addition to the standard UIF types. These are explained in Table 2-7.

Mapping CICS Data Types to UIF Data Types

The CICS connector has data types with the following attributes as shown in Table 2-2 through Table 2-6.

Table 2-2 CICS Attributes

Attributes	Description
Total_digits	Total number of digits including fractional part
Precision	Number of digits in the fractional part
Len	Length of the string field
User_type	These specify the data type

The four user types are explained in Table 2-3.

Table 2-3 User Types

User Type	Description
101 - zoned	The attribute total digits is needed to represent the length of the zoned value. Precision is optional.
102 - packed	The attribute total digits is needed to represent the length of the packed decimal. Precision is optional.
103 - short	Represents Short value (half word integer)
104 - long	Represents Long value (double word integer)

During runtime the UIF type is mapped to the EIS type. Table 2-4 describes the comparable UIF user type for each of the CICS primitive user types.

Table 2-4 Comparable UIF Primitive User Types for CICS User Types

Type in CICS	Type in UIF
Pic 9(n) Comp-4. or Pic 9(n) binary.	
short 1 =< n >= 4	T_integer + user_type = "103"
int 9 =< n >= 5	T_integer
long 10 =< n >= 18	T_double + user_type = "104"
Pic 9(n) Comp-1	T_float
Pic 9(n) Comp-2	T_double
Pic x(n)	T_fstring + size = "n"
Pic 9(n) Comp-3. or Pic 9(n) Packed -decimal	T-fstring + user_type = "102"
Pic 9(n)	T_fstring + user_type = "101"

The comparable types in UIF for CICS array types are described in Table 2-5.

Table 2-5 Comparable UIF Array User Types for CICS User Types

Type in CICS	Type in UIF
Fixed occurs length	T_array + array_size = "length"
Dynamic occursMinLength to MaxLength depending on <variable>	T_array + array_size="MaxLength"+ size_is = <variable>

The comparable types in UIF for CICS structure types are described in Table 2-6.

Table 2-6 Comparable UIF Structure User Types for CICS User Types

Type in CICS	Type in UIF
level structname	<T_struct name = "structname">
field level field 1 pic 9(4).	<T_integer name = "field1" user_type="103">
field level pic x(8).	<T_fstring name = "field2" size="8">

Table 2-6 Comparable UIF Structure User Types for CICS User Types

Type in CICS	Type in UIF
01 =< level <= 99	</T_struct>
level =< field level <= 99	

propertySet

The propertySet contains the properties of the operation.

The CICS propertySet contains one field: connection. It is used internally by the CICS connector. The value of this entry matches the function object name. See Figure 2-5.

Figure 2-5 PropertySet

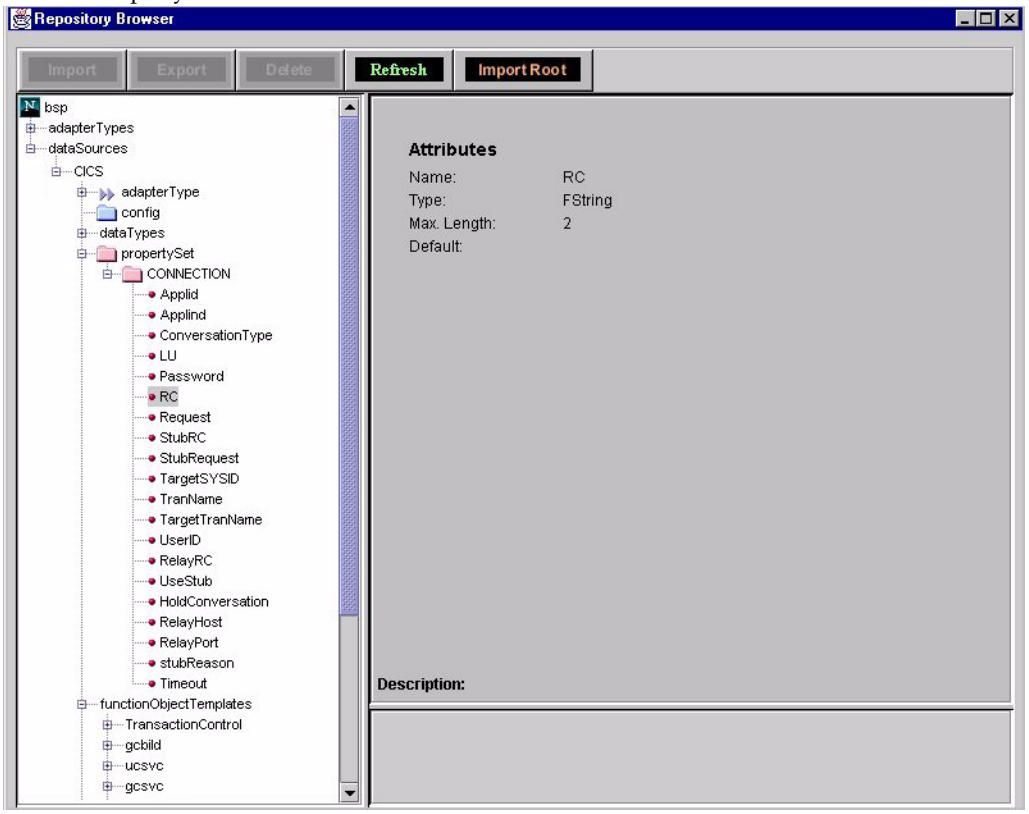


Table 2-7 lists the communication fields and includes a description.

Table 2-7 Description of Communication Fields

Field Name	Description
Applid	Specifies a CICS program name that is to be invoked on the CICS system.

Table 2-7 Description of Communication Fields *(Continued)*

Field Name	Description
Applind	Used to indicate the conversation state status, and can be in one of the following three states: E – Conversation is inactive. The conversation has either not yet started or already ended. The E state is an ending state. To leave the E state, the client application must start a new conversation. M – Client application has the right to send data and the service application must receive it. When in the M state the client application is in control of the conversation. S – Service application has the right to send data and the client application must receive it. When in the S state the service application is in control of the conversation. The client starts the client application from an inactive state (E). The client application's first call to iAS for CICS identifies the service application with all needed conversation parameters. The first call starts the service application on the remote platform, if not already started. The conversation then continues until one of the applications stops it, either normally, or abnormally, at which point the conversation enters the E state.
Conversation Type	Pooled — must be specified for TCP/IP connection. Direct — must be specified for APPC connection.
LU	The Logic Unit (LU) name where the transaction specified in TranName is defined if SRVXLINK is used. This is where SRVXLINK itself resides.

Table 2-7 Description of Communication Fields *(Continued)*

Field Name	Description
Request	This field specifies the communication operation that the Relay performs: The following listst the tpes of requests: SendAndEnd – Send data to the server application, wait for a reply, and then end the conversation. SendLast – Send data to the server application and wait for a reply. Send — Send data to the server application. Receive – Receive data from the server application. End – End the conversation with the server. Quit – Abort the conversation with the server. If you do not intend to use the Server Side API, you should use "SendLast" or "SendAndEnd".
StubRC	Return code from the SRVXLINK program.
StubRequest	Instructs SRVXLINK what to do. The following values are acceptable: COMMIT – to automatically commit the transaction upon successful execution of the EIS program. EXECUTE – not to automatically commit the transaction.
Target system ID	The system ID of the CICS in which SRVXLINK starts a program. If the program resides on the same CICS as SRVXLINK, no specification program is required.
Transaction Name	Default transaction name for the SRVXLINK program is SRVX.
Target Transaction Name	Not used in this version
Use Stub	Determines whether you are using the SRVXLINK program or not. Values should be “Y” or “N”.
Hold Conversation	This configuration places the conversation on “hold” for the user until he wants to access it again. This option guarantees that the user accesses the same CICS transaction on its next execution of an operation, within the same user interaction.
RC	Return Code. The return code is received by the CICS Connector during its conversation with the Relay.

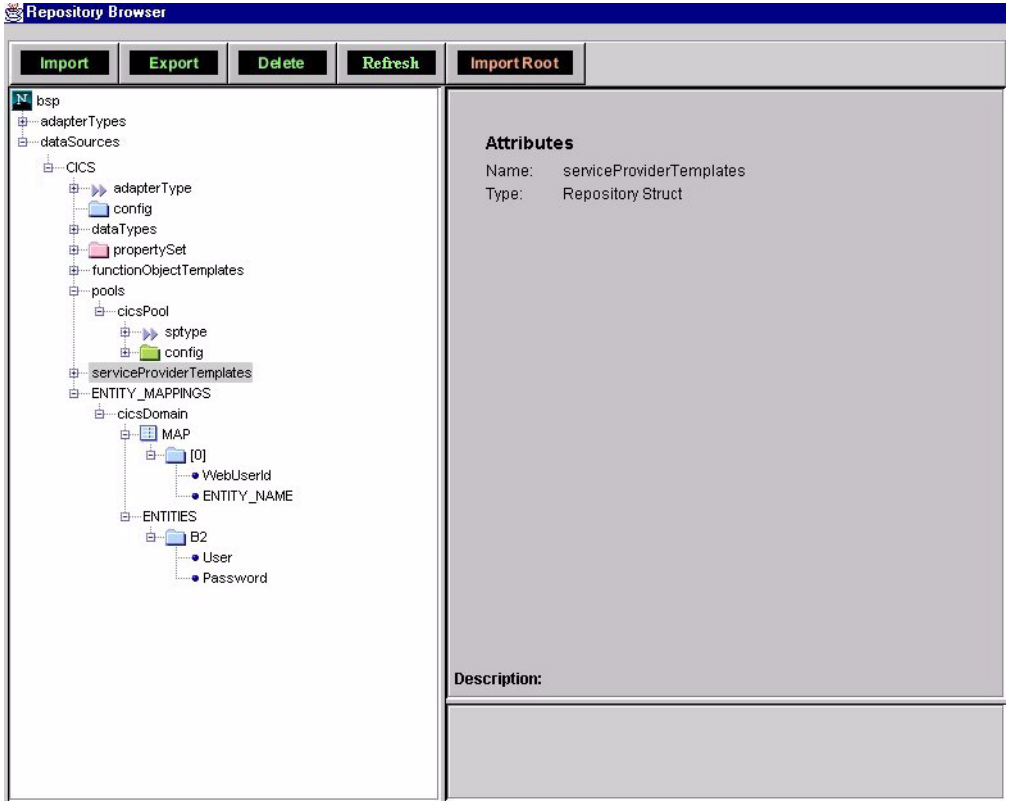
Table 2-7 Description of Communication Fields *(Continued)*

Field Name	Description
RelayRC	This is the return code (RC) that the Relay received from conversations with the CICS server.

Entity Mapping

User mapping information consist of definitions of user mapping tables from the web domain to EIS domain for a specific EIS. 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 CICS Administrator's Guide* for details on the Management Console.

Figure 2-6 Entity Mapping



The WebUserId represents the web domain and is mapped to the EIS domain.

Chapter 3

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 EIS. The UIF API is an object oriented framework.

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

The iPlanet Application Server Enterprise Connector for CICS allows you to access data through the data object interface.

The data object interface:

- presents a unified representation of EIS 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, see Figure 3-1 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, see Figure 3-2, 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"	

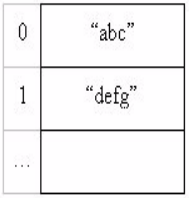
Circular references are not allowed. iPlanet Application Server Enterprise Connector for CICS prevents a data object being used as an attribute of itself. Indirect circular references are not checked.

CAUTION An error message is not generated if an indirect circular reference is defined. Unpredictable results occur if a circular reference is used at runtime.

Array Objects

An array object, see Figure 3-3, 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 fields corresponding data types see Table 3-1. 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.

Table 3-1 Type Information Objects

DataObjectInfo Type	Target Object
IBSPDataObjectPrimitiveInfo describes the type number, size of value (if type is string or binary), and the default value.	IBSPDataObjectPrimitive
IBSPDataObjectStructureInfo describes the type information of all fields of the target structure. The type information of each field is in turn described by a type information 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 information 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 primitive object type from integer to 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 an EIS. 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("CICS", "CICS_sp1");
    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
CICS	Data source name
CICS_sp1	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 CICS, 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(“CICS”, “phonebook”);
     }
}
```

Table 3-3 defines the function object parameters.

Table 3-3 Function Object Parameters

Parameter	Definition
CICS	Data source name
phonebook	Function name

Setting Up and Executing the Function Object

To Set Up and Execute the Function Object

1. Specify and enable the service provider.

2. Set the WebUserId that represents the web domain and that is mapped to the EIS domain.

   Refer to "CICS User Management", for details.

3. Prepare the function object, set up the propertySet, and set the input parameters in the function object's dataBlock.
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

```
private dataRecord executePB(IBSPRuntime runtime, IBSPServiceProvider sp,
          String requestCode, dataRecord input)   throws BspException
{
   int hr = 1;
   IBSPFunctionObject fn = null;
   IBSPDataObject data = null, prop = null;
   String lastname = new String("");
   String firstname = new String("");
   String extension = new String("");
   String zipcode = new String("");
   dataRecord o_entry = null;

   if( (runtime != null) && (sp != null) )
   {
      fn = runtime.createFunctionObject("CICS", "phonebook");
      hr = sp.enable();
      if( fn != null )
      {
         hr = fn.useServiceProvider(sp);
         hr = fn.prepare(requestCode);

         data = fn.getDataBlock();
         if( data != null )
         {
            data.setAttrFString("INPUT.REQC", requestCode);
            data.setAttrFString("INPUT.LNAME",input.m_lastName);
            data.setAttrFString("INPUT.FNAME",input.m_firstName);
            data.setAttrFString("INPUT.EXTENTION",input.m_extension);
            data.setAttrFString("INPUT.ZIPC",input.m_zipcode);

            prop = fn.getProperties();

            if( prop != null )
            {

               prop.setAttrInt("CONNECTION.StubReason",0 );
```

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

```
                }

                hr = fn.execute();
                if( hr == 0 )
                {
                    data = fn.getDataBlock();
                    if( data != null )
                    {
                        deb.println("MSG:
"+data.getAttrFString("OUTPUT.MSG"));
                        deb.println("REQC:
"+data.getAttrFString("OUTPUT.REQC"));
                        lastname = data.getAttrFString("OUTPUT.LNAME");
                        deb.println("LNAME: "+lastname);
                        firstname = data.getAttrFString("OUTPUT.FNAME");
                        deb.println("FNAME: "+firstname);
                        extension = data.getAttrFString("OUTPUT.EXTENTION");
                        deb.println("EXTENTION: "+extension);
                        zipcode = data.getAttrFString("OUTPUT.ZIPC");
                        deb.println("ZIPC: "+zipcode);
                        o_entry = new dataRecord(lastname, firstname,
extension, zipcode);
                        o_entry.m_message = new
String(data.getAttrFString("OUTPUT.MSG"));
                    }

                    prop = fn.getProperties();
                    if(prop != null)
                    {
                        deb.println("APPLID:
"+prop.getAttrFString("CONNECTION.Applid"));
                        deb.println("LU:
"+prop.getAttrFString("CONNECTION.LU"));
...

                    }
                } // if(hr == 0)
                else
                    deb.println("Search Failed");
            } // if(data != null)
            else
                deb.println("Search Failed");

          } // if( fn != null )
          else
              deb.println("Search Failed");
          hr = sp.disable();

       } // if( (runtime != null) && (sp != null) )
       else
          deb.println("Search Failed");
       return o_entry;
   }
```

CICS User Management

The application programmer provides a WebUserId to the CICS Enterprise connector, which determines the CICS 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("CICS", "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 Deploy

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 CICS 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 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 an 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 works no matter where you build the archive. For example, you can successfully deploy `.ear` files that you have built on NT 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 the iPlanet Application Server. For example, static HTML files,

Using the Command Line to Deploy

A Web Application Module can be deployed as a standalone 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 Examples

You must know how to acquire UIF objects and execute functions by using function objects to program for the iPlanet Application Server Enterprise Connector for CICS. You also need to know how to use servlets with the iPlanet Application Server Enterprise Connector for CICS to access functions on a CICS EIS.

This chapter includes the following information:

- CICS Sample
- Code Samples
- Telco Sample

CICS Sample

The CICS samples provided show the general flow of a connector program.

Activation

The CICS sample consist of servlets which activate CICS programs that access the CICS EIS.

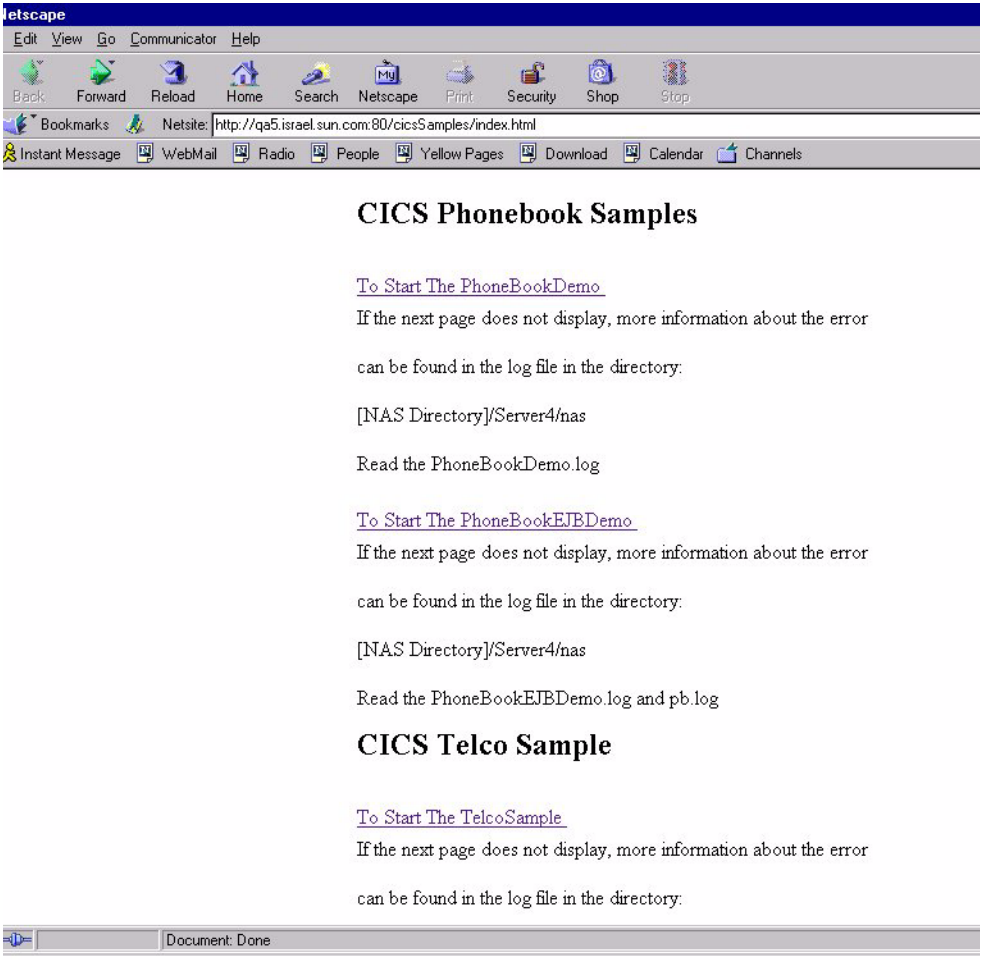
You must have set up the environment before activating the samples. See Post Installation Issues in Chapter 2 of the *iPlanet Application Server Enterprise Connector for CICS Administrator's Guide*.

To Run the CICS Samples on NT/2000

1. Select Programs>iPlanet Application Server 6.5 > CICS Connector 6.5 - Sample Applications.

   The CICS samples page is displayed with links to the samples.

2. Click on the link “To Start The PhoneBook Demo” to activate the PhoneBook servlet.



To Run the CICS Sample on Solaris

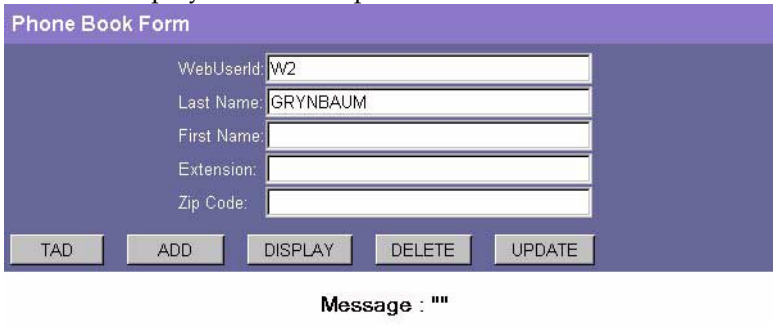
1. Start your browser.
2. Enter the following URL: <host name>:<web server port>/cicsSamples.

   The CICS Samples page appears, showing links to the Phone Book and Telco samples.

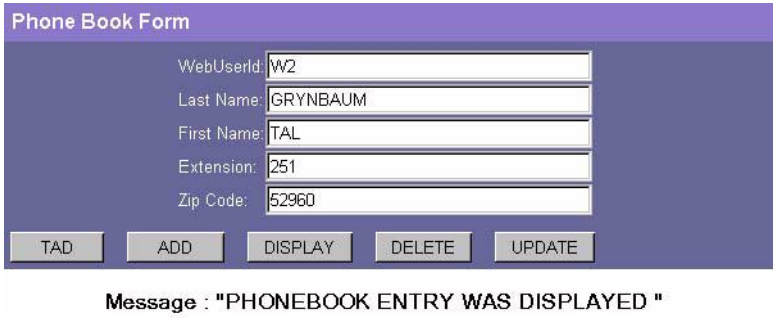
Phone Book Sample Operation

1. Click on: `To Start The PhoneBookDemo` link

   The Phone Book Form dialog box is displayed.
2. Type in the WebUserId and last name of the person for whom you want information.
3. Click Display to show the person's details.



4. The following display confirms the successful installation of the CICS connector.



Code Samples

The code samples are included to show the programmer how to set INPUT parameters, set the transaction name and get the OUPUT parameters.

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

PhoneBook Servlet Java Code

Code Example 4-1 PhoneBookServlet.java

```
/*
 * @(#)SimpleServlet.java1.22 97/10/25
 *
 * Copyright (c) 1996-1997 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information").  You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE
 * SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
 * PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES
 * SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 *
 * CopyrightVersion 1.0
 */
package PhoneBookDemo;

import java.io.*;
import com.kivasoft.IContext;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import javax.naming.*;
import javax.naming.spi.*;
import netscape.bsp.*;
import netscape.bsp.runtime.*;
import netscape.bsp.dataobject.*;
import netscape.bsp.BspException.*;
//   <form action="/NASApp/application name/PhoneBookServlet" method=get>

/**
```

Code Example 4-1 PhoneBookServlet.java

```
 * This is a simple example of an HTTP Servlet.  It responds to the GET
 * and HEAD methods of the HTTP protocol.
 */
public class PhoneBookServlet extends HttpServlet
{

    private TestDebug deb = new TestDebug("PhoneBookDemo.log");
    private Context _context;

    protected String getInputString(HttpServletRequestrequest, String
parameterName)
    {
        // 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();
        }
        return parameter;
    }

    Context getContext()
        throws Exception, BspException
    {

        Hashtable env = new Hashtable(11);
        env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.kivasoft.eb.jndi.GDSInitContextFactory");

        if (_context == null)
        {
            _context = new InitialContext(env );
        }
        return _context;
    }

    private void Do(IBSPRuntime runtime, IBSPServiceProvider sp, String
requestCode)  throws BspException
    {
        int hr = 1;
        IBSPFunctionObject fn = null;
        IBSPDataObject data = null, prop = null;

        deb.println("Before Do()");
        deb.println("TransactionControl");

        if( (runtime != null) && (sp != null) )
        {
            deb.println("Before createFunctionObject()");
```

Code Example 4-1 PhoneBookServlet.java

```
            fn = runtime.createFunctionObject("CICS", "TransactionControl");
            deb.println("After createFunctionObject()");
            hr = sp.enable();
            deb.println("After enable(), hr = "+hr);
            if( fn != null )
            {
               hr = fn.useServiceProvider(sp);
               deb.println("After useServiceProvider(), hr = "+hr);

               hr = fn.prepare(requestCode);
              prop = fn.getProperties();

               if( prop != null )
               {
                //prop.setAttrFString("CONNECTION.ConversationType", "POOLED");
               //  prop.setAttrFString("CONNECTION.Applid", "PHONBOOX");
               //  prop.setAttrFString("CONNECTION.Request", "SendAndEnd");
                   // prop.setAttrFString("CONNECTION.LU", "A06CICS1");
               //  prop.setAttrFString("CONNECTION.TranName", "SRVX");
                   // prop.setAttrFString("CONNECTION.RelayHost", "MF");
                   // prop.setAttrFString("CONNECTION.RelayPort", "4701");
                   // prop.setAttrFString("CONNECTION.TargetTranName", " ");
                   // prop.setAttrFString("CONNECTION.TargetSYSID", " ");
                   // prop.setAttrFString("CONNECTION.RelayRC", "00");
            //     prop.setAttrFString("CONNECTION.UseStub", "Y");
            //     prop.setAttrFString("CONNECTION.HoldConversation", "Y");
                   prop.setAttrInt("CONNECTION.StubReason",0 );

                   deb.println("Before execute()");
                   hr = fn.execute();
                   deb.println("After execute(), hr = "+hr);
                   if( hr == 0 ) // ??????????????????????
                   {
                      prop = fn.getProperties();

                      deb.println("After getProperties()");
                      if(prop != null)
                      {
                         deb.println(" * * * * * * ");
                         deb.println("APPLID:
"+prop.getAttrFString("CONNECTION.Applid"));
                         // deb.println("UserID:
"+prop.getAttrFString("CONNECTION.UserID"));
                         // deb.println("Password:
"+prop.getAttrFString("CONNECTION.Password"));
                         deb.println("LU:
"+prop.getAttrFString("CONNECTION.LU"));
                         deb.println("TargetSYSID:
"+prop.getAttrFString("CONNECTION.TargetSYSID"));
                         // deb.println("RelayHost:
"+prop.getAttrFString("CONNECTION.RelayHost"));
```

Code Example 4-1 PhoneBookServlet.java

```
                        // deb.println("RelayPort:
"+prop.getAttrFString("CONNECTION.RelayPort"));
                        deb.println("TranName:
"+prop.getAttrFString("CONNECTION.TranName"));
                        deb.println("TargetTranName:
"+prop.getAttrFString("CONNECTION.TargetTranName"));
                        deb.println("Request:
"+prop.getAttrFString("CONNECTION.Request"));
                        deb.println("Applind:
"+prop.getAttrFString("CONNECTION.Applind"));
                        deb.println("RC:
"+prop.getAttrFString("CONNECTION.RC"));
                        deb.println("RelayRC:
"+prop.getAttrFString("CONNECTION.RelayRC"));
                        deb.println("StubRC:
"+prop.getAttrFString("CONNECTION.StubRC"));
                        deb.println("StubReason:
"+Integer.toString(prop.getAttrInt("CONNECTION.")));
                        deb.println("StubRequest:
"+prop.getAttrFString("CONNECTION.StubRequest"));
                        deb.println("UseStub:
"+prop.getAttrFString("CONNECTION.UseStub"));
                        deb.println(" * * * * * * ");
                      }
                   } // if(hr == 0)
                   else
                      deb.println("Search Failed");
               } // if(prop != null)
               else
                  deb.println("Search Failed");
            } // if( fn != null )
            else
               deb.println("Search Failed");
            hr = sp.disable();
            deb.println("After disable(), hr = "+hr);

        } // if( (runtime != null) && (sp != null) )
        else
            deb.println("Search Failed");
        deb.println(" ");
        return;
    }

    private dataRecord executePB(IBSPRuntime runtime, IBSPServiceProvider sp,
              String requestCode, dataRecord input)   throws BspException
    {
        int hr = 1;
        IBSPFunctionObject fn = null;
        IBSPDataObject data = null, prop = null;
        String lastname = new String("");
        String firstname = new String("");
```

Code Example 4-1 PhoneBookServlet.java

```
        String extension = new String("");
        String zipcode = new String("");
        dataRecord o_entry = null;

        deb.println("Before executePB()");
        deb.println("PhoneBook");

        if( (runtime != null) && (sp != null) )
        {
           deb.println("Before createFunctionObject()");
           fn = runtime.createFunctionObject("CICS", "phonebook");
           deb.println("After createFunctionObject()");
           hr = sp.enable();
           deb.println("After enable(), hr = "+hr);
           if( fn != null )
           {
              hr = fn.useServiceProvider(sp);
              deb.println("After useServiceProvider(), hr = "+hr+" requestCode
: "+requestCode);

              hr = fn.prepare(requestCode);
              deb.println("After prepare(), hr = "+hr);

              data = fn.getDataBlock();
              deb.println("After getDataBlock()");
              if( data != null )
              {
                 //           data.setAttrFString("CICS.INPUT.BDATA","");
                 data.setAttrFString("INPUT.REQC", requestCode);
                 data.setAttrFString("INPUT.LNAME",input.m_lastName);
                 data.setAttrFString("INPUT.FNAME",input.m_firstName);
                 data.setAttrFString("INPUT.EXTENTION",input.m_extension);
                 data.setAttrFString("INPUT.ZIPC",input.m_zipcode);

                 prop = fn.getProperties();
                 deb.println("After getProperties()");

                 if( prop != null )
                 {
                    //prop.setAttrFString("CONNECTION.ConversationType",
"POOLED");
                    //prop.setAttrFString("CONNECTION.Applid", "PHONBOOX");
                    //prop.setAttrFString("CONNECTION.Request", "SendAndEnd");
                    // prop.setAttrFString("CONNECTION.LU", "A06CICS1");
                    //prop.setAttrFString("CONNECTION.TranName", "SRVX");
                    // prop.setAttrFString("CONNECTION.RelayHost", "MF");
                    // prop.setAttrFString("CONNECTION.RelayPort", "4701");
                    // prop.setAttrFString("CONNECTION.TargetTranName", " ");
                    // prop.setAttrFString("CONNECTION.TargetSYSID", " ");
                    // prop.setAttrFString("CONNECTION.RelayRC", "00");
                  //prop.setAttrFString("CONNECTION.UseStub", "Y");
                  //prop.setAttrFString("CONNECTION.HoldConversation", "Y");
                    prop.setAttrInt("CONNECTION.StubReason",0 );
```

Code Example 4-1 PhoneBookServlet.java

```
                    }

                    deb.println("Before execute()");
                    hr = fn.execute();
                    deb.println("After execute(), hr = "+hr);
                    if( hr == 0 )   // ??????????????????????
                    {
                        data = fn.getDataBlock();
                        deb.println("After getDataBlock()");
                        if( data != null )
                        {
                            deb.println(" * * * * * * ");
                            deb.println("MSG:
"+data.getAttrFString("OUTPUT.MSG"));
                            deb.println("REQC:
"+data.getAttrFString("OUTPUT.REQC"));
                            lastname = data.getAttrFString("OUTPUT.LNAME");
                            deb.println("LNAME: "+lastname);
                            firstname = data.getAttrFString("OUTPUT.FNAME");
                            deb.println("FNAME: "+firstname);
                            extension = data.getAttrFString("OUTPUT.EXTENTION");
                            deb.println("EXTENTION: "+extension);
                            zipcode = data.getAttrFString("OUTPUT.ZIPC");
                            deb.println("ZIPC: "+zipcode);
                            o_entry = new dataRecord(lastname, firstname,
extension, zipcode);
                            o_entry.m_message = new
String(data.getAttrFString("OUTPUT.MSG"));
                        }

                        prop = fn.getProperties();
                        deb.println("After getProperties()");
                        if(prop != null)
                        {
                            deb.println("APPLID:
"+prop.getAttrFString("CONNECTION.Applid"));
                            // deb.println("UserID:
"+prop.getAttrFString("CONNECTION.UserID"));
                            // deb.println("Password:
"+prop.getAttrFString("CONNECTION.Password"));
                            deb.println("LU:
"+prop.getAttrFString("CONNECTION.LU"));
                            deb.println("TargetSYSID:
"+prop.getAttrFString("CONNECTION.TargetSYSID"));
                            // deb.println("RelayHost:
"+prop.getAttrFString("CONNECTION.RelayHost"));
                            // deb.println("RelayPort:
"+prop.getAttrFString("CONNECTION.RelayPort"));
                            deb.println("TranName:
"+prop.getAttrFString("CONNECTION.TranName"));
```

Code Example 4-1 PhoneBookServlet.java

```
                        deb.println("TargetTranName:
"+prop.getAttrFString("CONNECTION.TargetTranName"));
                        deb.println("Request:
"+prop.getAttrFString("CONNECTION.Request"));
                        deb.println("Applind:
"+prop.getAttrFString("CONNECTION.Applind"));
                        deb.println("RC:
"+prop.getAttrFString("CONNECTION.RC"));
                        deb.println("RelayRC:
"+prop.getAttrFString("CONNECTION.RelayRC"));
                        deb.println("StubRC:
"+prop.getAttrFString("CONNECTION.StubRC"));
                        deb.println("StubReason:
"+Integer.toString(prop.getAttrInt("CONNECTION.StubReason")));
                        deb.println("StubRequest:
"+prop.getAttrFString("CONNECTION.StubRequest"));
                        deb.println("UseStub:
"+prop.getAttrFString("CONNECTION.UseStub"));
                        deb.println(" * * * * * * ");
                    }
                } // if(hr == 0)
                else
                    deb.println("Search Failed");
            } // if(data != null)
            else
                deb.println("Search Failed");

        } // if( fn != null )
        else
            deb.println("Search Failed");
        hr = sp.disable();
        deb.println("After disable(), hr = "+hr);

    } // if( (runtime != null) && (sp != null) )
    else
        deb.println("Search Failed");
    deb.println(" ");
    return o_entry;
}

private IBSPRuntime getRuntime()   throws BspException
{
    com.kivasoft.IContext _ctx =
        ((com.netscape.server.servlet.platformhttp.PlatformServletContext)
        getServletContext()).getContext();

    deb.println(" before access_cBSPRuntime.getcBSPRuntime ");
    IBSPRuntime ibspruntime = access_cBSPRuntime.getcBSPRuntime(_ctx, null,
null);
    deb.println(" after access_cBSPRuntime.getcBSPRuntime ");
    return ibspruntime;
}
```

Code Example 4-1 PhoneBookServlet.java

```
    private IBSPServiceProvider getServiceProvider(IBSPRuntime runtime)
    {
       deb.println("Before createServiceProvider()");
       if (runtime != null)
          return runtime.createServiceProvider("CICS", "CICS_sp1");
       else
          deb.println("runtime is null");
       return null;
    }

    public dataRecord sendRequest(String i_requestCode, dataRecord
i_entry,String WebUserId) throws Exception
    {
       dataRecord output = null;
       int operation = -1;
   IBSPServiceProvider sp=null;
   int hr=1;

   try
   {
       deb.println("i_requestCode : "+i_requestCode);
       deb.println(" Before getRuntime()");
       IBSPRuntime runtime = getRuntime();
       deb.println("After getRuntime()");
       if( runtime != null )
       {
          deb.println("Before getServiceProvider()");
          sp = getServiceProvider(runtime);
          deb.println("After getServiceProvider()");
          if( sp != null )
          {
       IBSPDataObject config=sp.getConfig();


       if(WebUserId!=null)


       {
          config.setAttrFString("WebUserId",WebUserId);
          deb.println("after set webuserid"+WebUserId);


        }
             deb.println("requestCode : "+i_requestCode);
             if (i_requestCode.equals("TAD") ||
                i_requestCode.equals("ADD") ||
                i_requestCode.equals("DISPLAY") ||
                i_requestCode.equals("DELETE") ||
                i_requestCode.equals("UPDATE"))
                output = executePB(runtime,sp, i_requestCode, i_entry);

             if (i_requestCode.equals("PREPARE") ||
                i_requestCode.equals("ROLLBACK") ||
```

Code Example 4-1 PhoneBookServlet.java

```
                i_requestCode.equals("COMMIT"))
                Do(runtime, sp,  i_requestCode);
          }

       }
 }
 catch(BspException BspError)
{
   if(sp!=null)
   {
     hr = sp.disable();
          deb.println("After disable(), hr = "+hr);
   }
   throw(BspError);
}
catch(Exception Error)
{
   if(sp!=null)
   {
     hr = sp.disable();
          deb.println("After disable(), hr = "+hr);
   }
   throw(Error);
}
      return (output);
  }

  /**
   * Handle the GET and HEAD methods by building a simple web page.
   * HEAD is just like GET, except that the server returns only the
   * headers (including content length) not the body we write.
   */
  public void doGet (
      HttpServletRequestrequest,
      HttpServletResponseresponse ) throws ServletException, IOException
  {
      deb.println("4. Before sendRequest ");
 RequestDispatcher dispatcher=null;
 try
 {
     String WebUserId=getInputString(request,"WebUserId");

     dataRecord input = new dataRecord(
            getInputString(request,"lastname"),
            getInputString(request,"firstname"),
            getInputString(request,"extension"),
            getInputString(request,"zipcode"));
     deb.println("4a. last Name : "+ input.m_lastName);
     String oper = null;
     oper = getInputString(request,"tad");
     if(oper == null)
        oper = getInputString(request,"add");
     if(oper == null)
        oper = getInputString(request,"disp");
```

Code Example 4-1 PhoneBookServlet.java

```
        if(oper == null)
           oper = getInputString(request,"del");
        if(oper == null)
           oper = getInputString(request,"update");
        if(oper == null)
           oper = getInputString(request,"prepare");
        if(oper == null)
           oper = getInputString(request,"rollback");
        if(oper == null)
           oper = getInputString(request,"commit");
        deb.println("5. Operation:"+oper);

        dataRecord output = sendRequest(oper,input,WebUserId);
        deb.println("6. After sendRequest ");
        deb.println("7. first Name : "+ output.m_firstName);
        deb.println("8. extension : "+ output.m_extension );
        deb.println("9. zipcode: "+ output.m_zipcode );

    response.setContentType("text/html");
        dispatcher =
getServletContext().getRequestDispatcher("/jsp/PhoneBookForm.jsp");
        request.setAttribute("lastname_value",output.m_lastName);
        request.setAttribute("firstname_value",output.m_firstName);
        request.setAttribute("extension_value",output.m_extension);
        request.setAttribute("zipcode_value",output.m_zipcode);
        request.setAttribute("message_value",output.m_message);

    }
    catch(BspException BspError)
       {
        deb.println("BspException:"+BspError.getMessage());
     response.setContentType("text/html");
        dispatcher =
getServletContext().getRequestDispatcher("/jsp/ExceptionForm.jsp");

     IBSPDataObjectStructure info = null;
         info=BspError.getInfo();


         if (info != null && info.attrExists("msgid") )



request.setAttribute("BspException",BspError.getMessage()+",Error code : " +
info.getAttrInt("msgid"));
     else
            request.setAttribute("BspException",BspError.getMessage());
       }
    catch(Exception exception)
       {
        deb.println("Exception:"+exception.toString());
     response.setContentType("text/html");
```

Code Example 4-1 PhoneBookServlet.java

```
       dispatcher =
getServletContext().getRequestDispatcher("/jsp/ExceptionForm.jsp");
       request.setAttribute("Exception",exception.toString());
      }
      dispatcher.include(request, response);

    deb.println("8. before return ");
    return;
     }



     /** handles the HTTP POST operation **/
     public void doPost (HttpServletRequest request,HttpServletResponse
response)
        throws ServletException, IOException {
        doGet(request,response);
     }

}
```

PhoneBook JSP Sample Code

Code Example 4-2 PhoneBookForm.jsp

```
<%
     String lastname_value = "GRYNBAUM";
     /** get the error message from the request attributes **/
     Object obj1 = request.getAttribute("lastname_value");
     if (obj1!=null) {
         lastname_value = (String) obj1;
     }
%>
<%
     String firstname_value = "";
     /** get the error message from the request attributes **/
     Object obj2 = request.getAttribute("firstname_value");
     if (obj2!=null) {
         firstname_value = (String) obj2;
     }
%>
<%
     String extension_value = "";
     /** get the error message from the request attributes **/
     Object obj3 = request.getAttribute("extension_value");
     if (obj3!=null) {
         extension_value = (String) obj3;
```

Code Example 4-2 PhoneBookForm.jsp

```
    }
%>
<%
    String zipcode_value = "";
    /** get the error message from the request attributes **/
    Object obj4 = request.getAttribute("zipcode_value");
    if (obj4!=null) {
       zipcode_value = (String) obj4;
    }
%>
<%
    String message_value = "";
    /** get the error message from the request attributes **/
    Object obj5 = request.getAttribute("message_value");
    if (obj5!=null) {
       message_value = (String) obj5;
    }
%>
<%
    String WebUserId = "";
    /** get the error message from the request attributes **/
    Object obj6 = request.getParameter("WebUserId");
    if (obj6!=null) {
       WebUserId = (String) obj6;
    }
%>
<HTML>
<HEAD>
    <TITLE>Phone Book Demo</TITLE>
</HEAD>
<BODY>
<center>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600" >
    <tr>
    <td VALIGN=TOP WIDTH="420">
    <form action="/NASApp/PhoneBookDemo/PhoneBookServlet" method=post>
    <TABLE BORDER=0 CELLSPACING=0 CELLPADDING=6 WIDTH="100%">
       <tr>
          <td BGCOLOR="#8979C8">
             <b><font face="sans-serif,arial,helvetica" color=white>
             Phone Book Form</font></b>
          </td>
       </tr>
       <tr VALIGN=TOP>
          <td BGCOLOR="#666699">
          <p>
          <center>
          <TABLE BORDER=0 cellspacing=0 cellpadding=0>


          <tr>
```

Code Example 4-2 PhoneBookForm.jsp

```
              <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">Last Name:</font></td>
              <td><font face="arial, helvetica, sans-serif">
                 <input type="text" name="lastname"
value="<%=lastname_value%>" size=20 maxsize=50></font>
              </td>
           </tr>
           <tr>
              <td><font face="arial, helvetica, sans-serif" color=white
size="-1">First Name:</font></td>
              <td><font face="arial, helvetica, sans-serif">
                 <input type="text" name="firstname"
value="<%=firstname_value%>" size=20 maxsize=50></font>
              </td>
           </tr>
           <tr>
              <td><font face="arial, helvetica, sans-serif" color=white
size="-1">Extension:</font></td>
              <td><font face="arial, helvetica, sans-serif">
                 <input type="text" name="extension"
value="<%=extension_value%>" size=20 maxsize=20></font>
              </td>
           </tr>
           <tr>
              <td><font face="arial, helvetica, sans-serif" color=white
size="-1">Zip Code:</font></td>
              <td><font face="arial, helvetica, sans-serif">
                 <input type="text" name="zipcode" value="<%=zipcode_value%>"
size=20 maxsize=20></font>
              </td>
           </tr>
           <tr>
              <td><font face="arial, helvetica, sans-serif">
```

Code Example 4-2 PhoneBookForm.jsp

```
                  <input type="hidden" name="WebUserId"  value="<%=WebUserId%>"
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="tad" value="   TAD   ">
           
          </td>
          <td valign=top><input type="submit" name="add" value="   ADD   ">
           
          </td>
          <td valign=top><input type="submit" name="disp" value="DISPLAY">
           
          </td>
          <td valign=top><input type="submit" name="del" value="DELETE">
           
          </td>
          <td valign=top><input type="submit" name="update" value="UPDATE">
           
          </td>
          </tr>
       </TABLE>
       </td>
    </tr>
    </TABLE>
    </td>
    </tr>
    </form>
</TABLE>
<BR><B>Message : "<%=message_value%>" </B></BR>
<!--#include file="Copyright.jsp" -->
</BODY>
</HTML>
```

Telco Sample

Refer to the *iPlanet Application Server Enterprise Connector for CICS Administrator's Guide* for information on how to activate the CICS Telco sample.

Appendix A

Error Messages and Codes

The developer writes servlets that use connectors to connect with the EIS. 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 catch 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 to Relay or SRVXLINK: RC:{return code}, RelayRC:{relay return code}, StubRC:{stub return code}, StubReason:{stub error code}.
3	{Input\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 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	Failed to {get\set} user defined type.
17	Failed to initialize marshalling parameters, error code: {return code}.
18	Array {name} has {number} element,{text} {valid value} elements.
19	Illegal value={value} of field {field name}.
20	Field {field name) has illegal Value {value} ,contain more digits than {max digits}.
21	Field {field name} has illegal Value {value} ,contain illegal character {char}.
22	Field {field name} has invalid length,Max Len {length}, is smaller than {property name} {value}.

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 CICS connector 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")
;
                }
}
```

Appendix B

Communication Failure Codes

This appendix describes the codes that are output if there is a communication failure. The communication error may occur during the execute method of the function object.

The CONNECTION structure in the properties data block contains communication status information in the following fields.

- CONNECTION.RC
- CONNECTION.RelayRC
- CONNECTION.StubRC
- CONNECTION.StubReason

A zero value in each of these fields indicates a successful communication process. If the communication process failed, one, or more, of the above fields contain a failure code that can be used to identify the source of the problem.

CONNECTION.RC

Table B-1 lists the most common causes for CONNECTION.RC errors.

Table B-1 CONNECTION.RC Errors

Error Code	Description and Action
05	Session not bound, allocation failed *Description*: Conversation can not be started because the session is not bound. This could be a configuration or operational problem. *Action*: If this connection has worked before, contact the network operator in order to activate the connection.

Table B-1 CONNECTION.RC Errors

Error Code	Description and Action
20	Transaction id error
	Description: The transaction id does not exist (mainframe only).
	Action: Check the application table to check whether the application exists or whether a default entry is present
43	Error on getting application record
	Description: A conversation state error has been encountered. The application has issued a request incompatible with the current state. This is probably due to a programming error.
	Action: Check the protocol according to protocol specifications.
81	Async: connection record not yet arrived
	Description: A connection record has been passed to the non-blocking interface but the result is not yet available. This happens when the non-blocking application tries to check whether a reply is available. This is not necessarily an error and should be handled by the application. The conversation continues.
	Action: The calling program must retry later with the same connection record.
82	Async: Timeout
	Description: A timeout occurs when communicating through the non-blocking interface. This is similar to error type 81.
	Action: The calling program must retry later with the same connection record.
83	Timeout - cleanup
	Description: A call to SRVRQT has timed out. The problem is probably caused by the remote application which is not responding. The session is cleaned up and the sockets are closed.
	Action: Correct the remote application or supply a larger timeout value.

Table B-1 CONNECTION.RC Errors

Error Code	Description and Action
91	Application Table Missing or Invalid
	Description: The application table is missing in the path specified by the SRVAPPL specification and in the working directory.
	Action: Specify the SRVAPPL environment variable to point to a valid application table or create a valid application table in your working directory.
92	Workstation communication error
	Description: the connection is lost during the conversation. The most probable cause for this is that the remote application has abended. If this is not the case, the session was deactivated by force.
	Action: Find the cause of the remote application abend and correct the problem. If the remote application was not abended check whether the connection was force deactivated.

CONNECTION.RelayRC

These are the same as those listed for CONNECTION.RC. in Table B-1.

CONNECTION.StubRC

The possible values for CONNECTION.StubReason are listed in Table B-2.

Table B-2 CONNECTION.StubReason codes

StubReason	Code
CICO_XL_SUCCESS	DC C'00'
CICO_XL_LINKERR	DC C'01'
CICO_XL_NOTPREPARED	DC C'02'
CICO_XL_ALREADYPREPARED	DC C'03'
CICO_XL_MUSTROLLBACK	DC C'04'
CICO_XL_COMMITFAILED	DC C'05'

Table B-2 CONNECTION.StubReason codes

StubReason	Code
CICO_XL_ROLLBACKFAILED	DC C′06′
CICO_XL_SIGNONFAILED	DC C′07′

CONNECTION.StubReason

Possible values for the CONNUNICATION.StubReason can be found in the CICS documentation.

Glossary

Application Programming Interface (API) The interface to a library of language-specific subroutines that implement higher level functions. A set of calling conventions defining how a service is invoked through a software package.

Applet A component that typically executes in a web browser, but can execute in a variety of other applications or devices that support the applet programming model.

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.

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 PeopleSoft 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.

iAS (iPlanet Application Server) The iAS provides a 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.

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 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 markup language that allows you to define the tags(markup) needed to identify the data and text in XML documents.

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