Developer's Guide

iPlanet Application Server Enterprise Connector for PeopleSoft

Version 6.5

806-5512-02
August 2002

Contents

List of Figures

List of Tables

List of Procedures

List of Code Examples

Preface

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

This preface contains information about the following topics:

- Prerequisites
- What's in This Guide
- Document 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
- PeopleSoft programming concepts
- Java programming language and Java servlets
- Enterprise JavaBeans

What's in This Guide

The iPlanet Application Server Enterprise Connector for PeopleSoft Developer's Guide covers the information you need to know to write servlets or EJBs using the Unified Integration Framework and the iPlanet Application Server Enterprise Connector for PeopleSoft to access the PeopleSoft back end.

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

Chapter	Description
Chapter 1, "Overview"	Concepts you should understand before you set up iPlanet Application Server Enterprise Connector for PeopleSoft or use the API.
Chapter 2, "Viewing the Repository Contents"	Contains the procedure for acquiring UIF objects and executing functions. Also, describes how to use the management console to view and change your Enterprise Connector for PeopleSoft configuration, load files into the repository, and edit files.
Chapter 3, "Working With Data Objects"	Contains the procedure for acquiring UIF objects and executing functions. Also, describes how to use the management console to view and change your Enterprise Connector for PeopleSoft configuration, load files into the repository, and edit files.
Chapter 4, "Programming Examples"	Gives a simple sample application and examples of code that can be used to set up the PeopleSoft connector. Contains details instructions for building the Message Definition that is needed to run the sample application.
Appendix A, "Error Messages"	This lists the error messages and their codes. It also includes a sample of the error handling code that should be used by the servlet developer.

Document 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 PeopleSoft 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

These are additional documents that are available:

The following lists the documents that are available:

- *iPlanet Application Server Enterprise Connector for PeopleSoft 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*
- *Unified Integration Framework (UIF) Release Notes*
- *Unified Integration Framework (UIF) Developer's Guide*)

Chapter 1

Overview

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

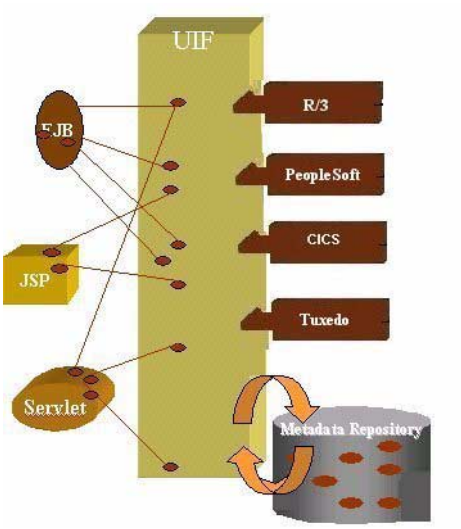
The following topics are covered in this chapter:

- Unified Integration Framework (UIF)
- The Three-tier Application Model
- Enterprise Connector Tools for PeopleSoft

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, see 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. iPlanet Application Server plays a prominent role in a three-tier application model. See the "The Three-tier Application Model" that follows 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:

- Run Time

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

Run Time

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 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.

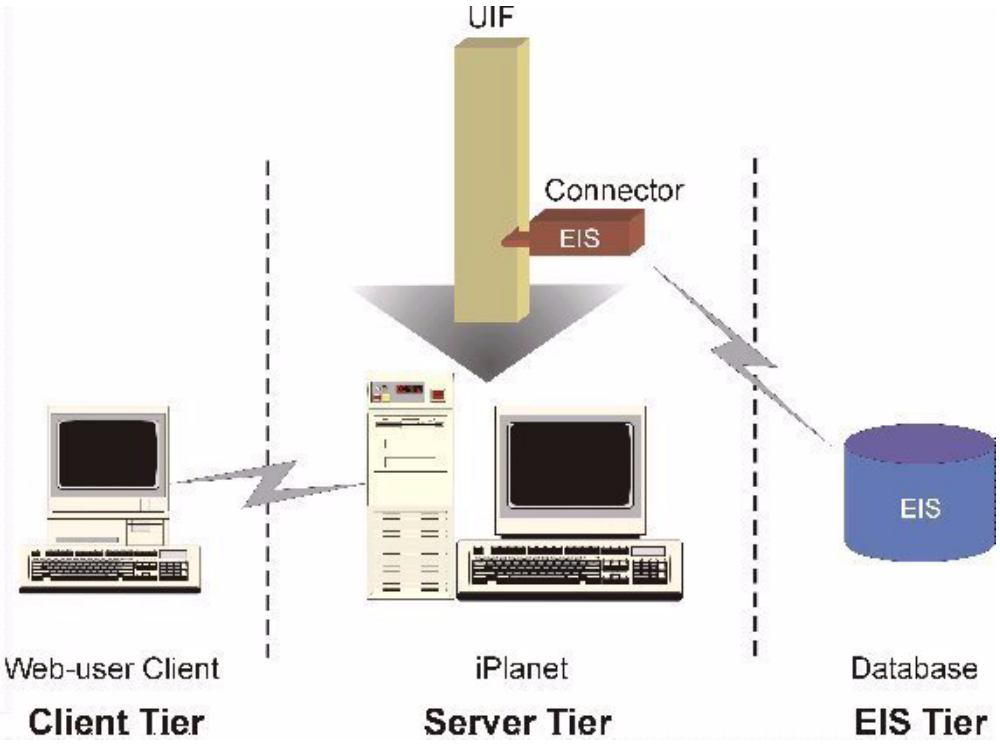
The Three-tier Application Model

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 iPlanet Application Server and enables data communication with diverse back-end EIS in a seamless and uniform manner.

Figure 1-2 The three tiers of web-based computing



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 (Back-end) Tier

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

Enterprise Connector Tools for PeopleSoft

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 back-end system, edit and manage data sources.

    A *data source* represents the connection to the back-end system.
  - Data Mining Tool - includes capabilities of 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 PeopleSoft 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 as a content viewer. 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 browser. XML files can be imported and exported using the browser but this 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.

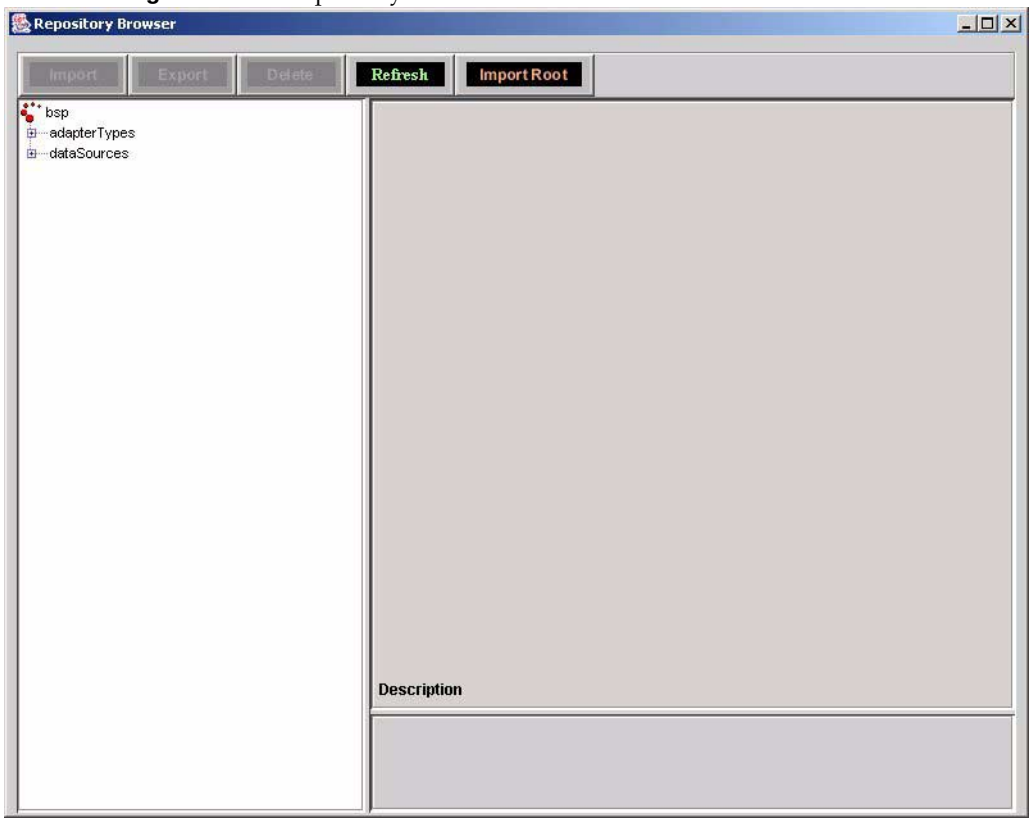
To Run the Repository Browser

- On Windows NT/2000 Select Programs>iPlanet Application Server 6.5>UIF 6.5 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 sides. When you open the browser. the left side 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 side shows the properties of the node selected on the left side.

View Hierarchy

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

- the root node
- connector types
- data sources

Refresh Display of Repository Contents

Click Refresh to refresh the display of the Repository contents.

View 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 backend 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

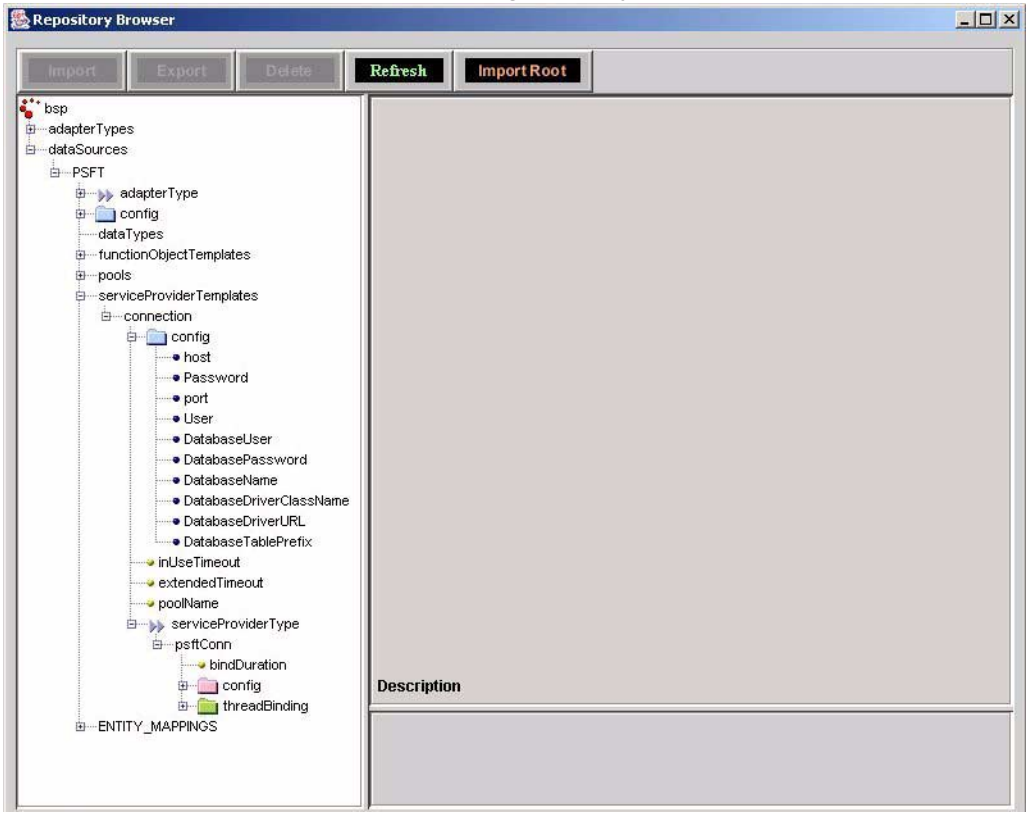


Table 2-1 lists the fields and the definitions of all the fields.

Table 2-1 Service Provider Configuration Object Field Definitions

Field	Definition
host	Host machine name of the target machine.
Password	PeopleSoft operator password.
port	Port that is used to connect to PeopleSoft.
User	PeopleSoft user name.
DatabaseUser	Used to log on for data mining.
DatabasePassword	Oracle password.

Table 2-1 Service Provider Configuration Object Field Definitions

Field	Definition
DatabaseName	The name of the database to which the user will attach. This parameter can be restricted. This parameter is not changeable when connecting to the application server.
DatabaseDriverClass Name	The database type, such as Oracle. This parameter can be restricted.
DatabaseDriver URL	
Database Table Prefix	Name of Oracle host machine.
DatabasePort	Name of port of Oracle database.

The pool name identifies the pool from which the service provider will provide the connection. This field value must be *dynamicPooled, meaning that the service provider can provide connection from every existing pool. The service provider will provide a connection from the pool that is applicable to the web domain.

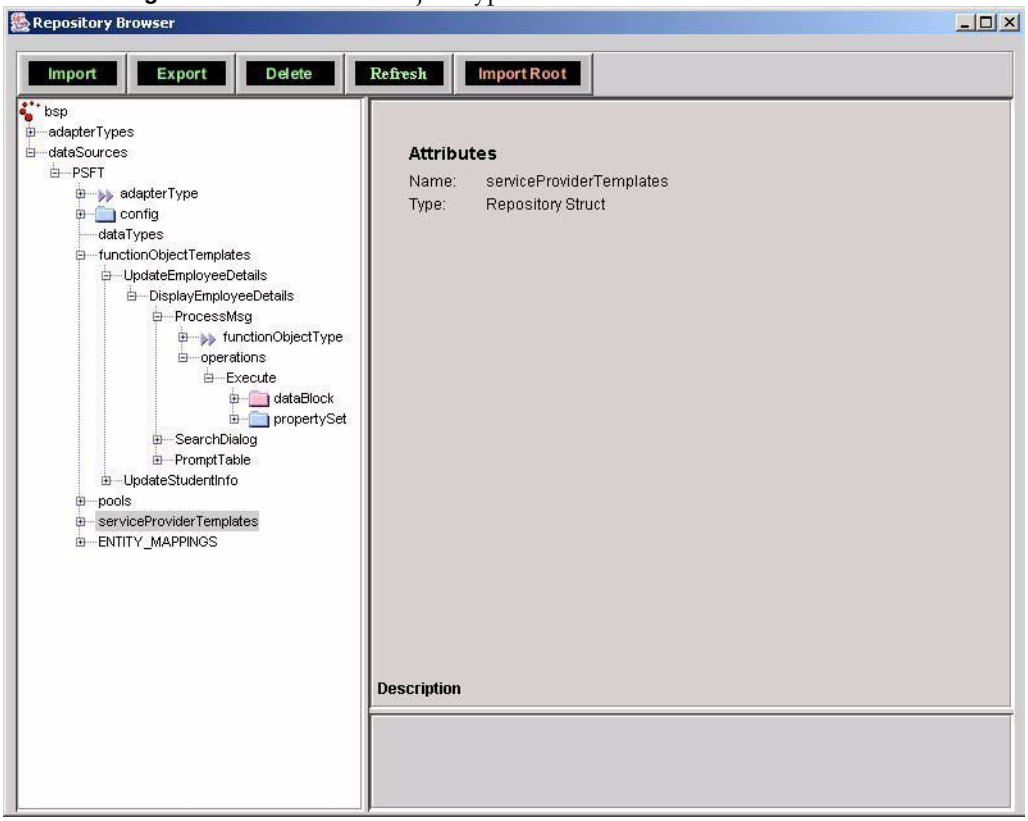
Every web user has a unique pool that is defined in the Entity mapping. See “Entity Mapping".

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.

Figure 2-3 Function Object Type

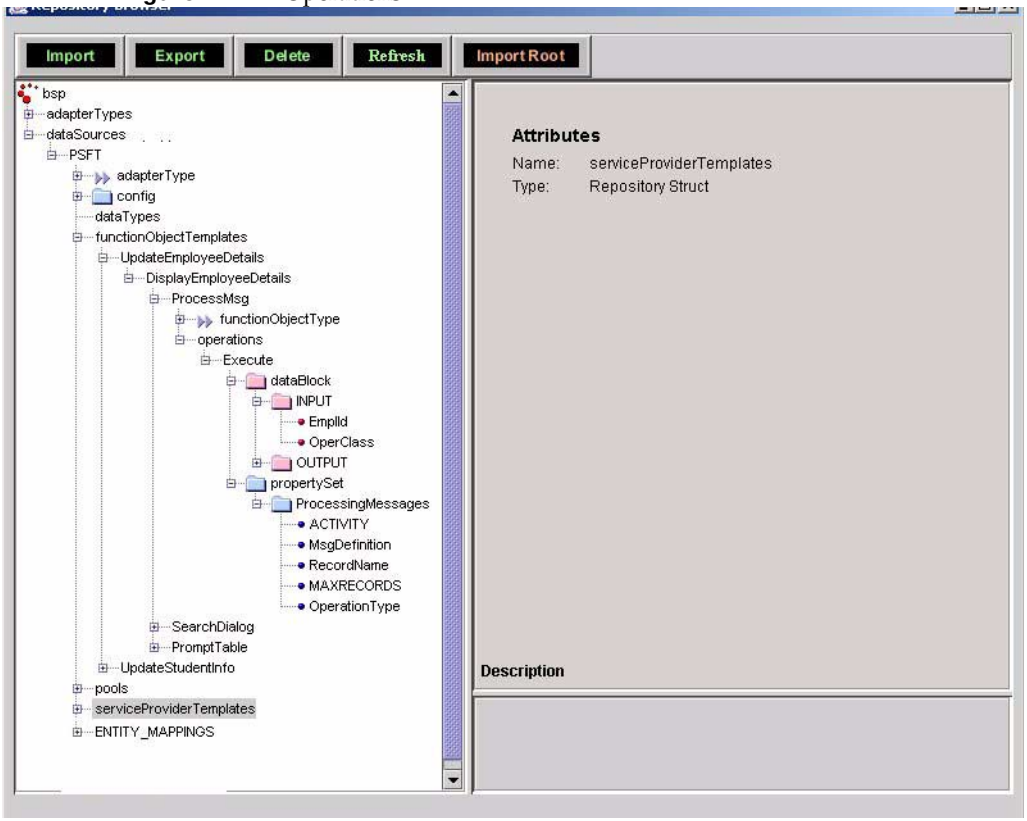


Operations

The PeopleSoft operations node has one operation: *Execute*.

The Execute node contain dataBlock and propertySets 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 length
- Default - default value that the field contains

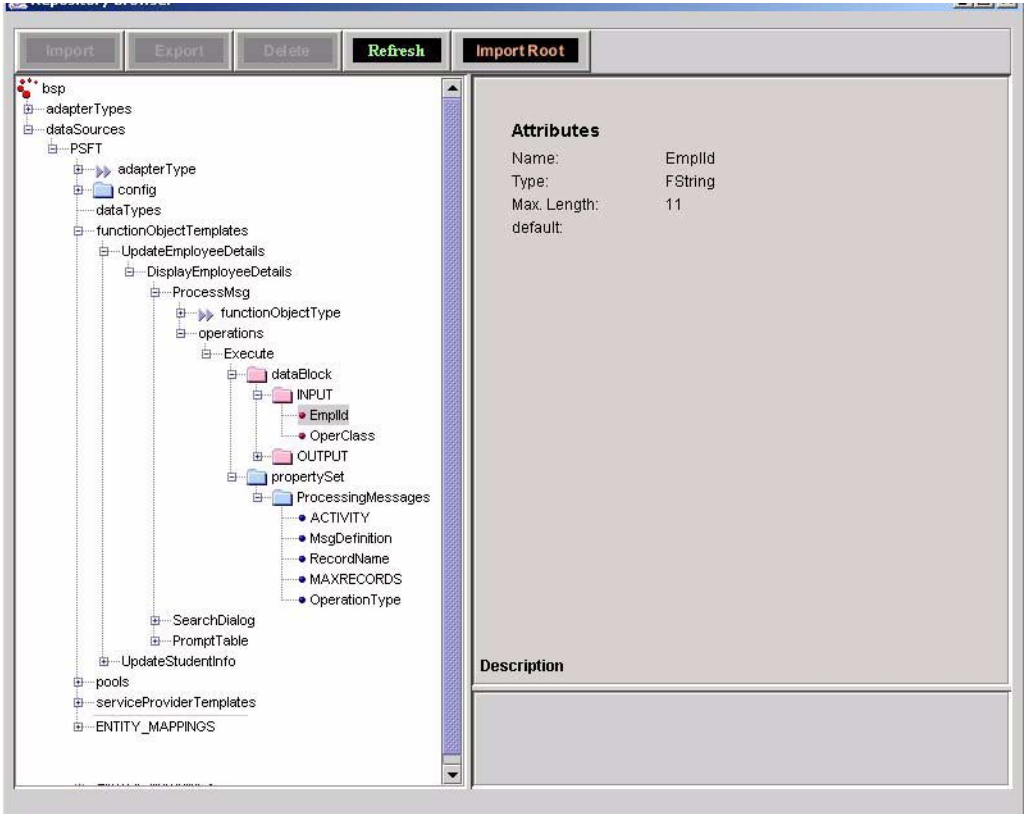
Other Field Attributes

Every connector has its own field types in addition to the nonstandard UIF types.

The PeopleSoft connector uses two field attributes for fields with decimals:

- Total_digits: total number of digits including fractional part
- Precision: number of digits in the fractional part

Figure 2-5 Field Attributes



These types are represented by the user-type.

Mapping PeopleSoft Data Types to UIF Data Types

The iPlanet Application Server Enterprise Connector for PeopleSoft 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 PeopleSoft and the equivalent type in UIF

Table 2-2 PeopleSoft User Types

Type in PeopleSoft	Type in UIF
Character	FString
Long Character	FString
Number	Integer, Double
Signed Number	Integer, Double
Date	FString
Time	FString
DateTime	FString

Property Set

The property set contains the properties of the operation.

The PeopleSoft property set contains the following fields:

- ACTIVITY
- MsgDefinition
- RecordName
- MAXRECORDS
- OperationType - This can have values of: 0 - Process Messages, 1 - Prompt Table, or 2 - Search Dialog.

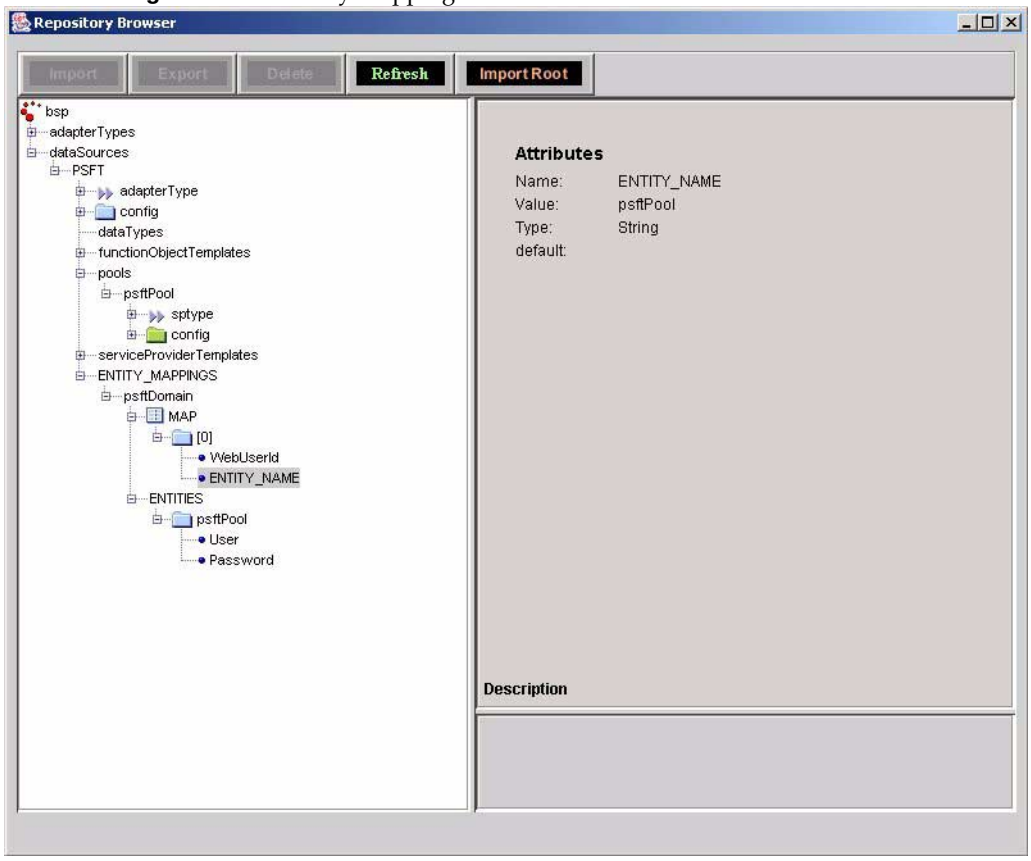
MAXRECORDS is the maximum number of rows to return. If you specify MAXRECORDS as 0, the Message Agent will use the row limit for your system (300 by default).

The other four fields are used by the connector. 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 PeopleSoft 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 backend 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 backend system. The UIF API is an object-oriented framework.

The iPlanet Application Server Enterprise Connector for PeopleSoft is used to execute PeopleSoft functions on a remote PeopleSoft server. The servlet or EJB uses the PeopleSoft connector to access the PeopleSoft 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 PeopleSoft allows you to access data through the data-object interface.

The data-object interface:

- presents a unified representation of backend data types
- represents complex data

- supports most common primitive data types.

The types of data objects are:

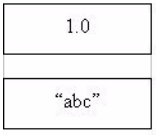
- Primitive Objects
- Structure Objects
- Array Objects
- Type Information Objects

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, in 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 PeopleSoft prevents a data object being used as an attribute of itself. Indirect circular references are not checked by iPlanet Application Server for PeopleSoft.

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 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.

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

NOTE The PeopleSoft connectors do not support image data types.

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 backend 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("PSFT", "connection");
     else
```

Code Example 3-3 Creating the Service Provider Object

```
        deb.println("runtime is null");
    return null;
}
```

Table 3-2 defines the function object parameters.

Table 3-2 Service Provider Object Types

Parameter	Definition
PSFT	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 PeopleSoft, 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()");
```

Code Example 3-4 Creating the Function Object

```
        fn = runtime.createFunctionObject(“PSFT”, “ProcessMsg”);
    }
}
```

Table 3-3 defines the function object parameters.

Table 3-3 Function Object Parameters

Parameter	Definition
PSFT	Data source name
ProcessMsg	PSFT 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 web-user ID that represents the web domain and will be mapped to the backend domain.
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;
```

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

```
int hr = 1;

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

        sp = getServiceProvider(runtime);
        if( sp != null )
        {
            IBSPDataObject config=sp.getConfig();
            config.setAttrFString("WebUserId",WebUserId);
            fn = runtime.createFunctionObject("PSFT", "ProcessMsg");
            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");
```

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

```
    }
    catch(BspException BspError)
    {
.
.
.
```

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.

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 PeopleSoft Administrator's Guide*, Chapter 3 - Managing Data: The Data Mining Tool.

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

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

The following sections describe the above mentioned methods of deployment.

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 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 will work no matter where you build the archive. For example, you can successfully deploy EAR files that you have built on Windows NT/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 iPlanet Application Server and then registered with iPlanet Application Server. For example, static HTML files,

Using the Command Line to Deploy

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 Examples

To program for iPlanet Application Server Enterprise Connector for PeopleSoft , you must know how to acquire Unified Integration Framework (UIF) objects and execute functions by using function objects. You also need to know how to use servlets with the iPlanet Application Server Enterprise Connector for PeopleSoft to access functions on a PeopleSoft EIS. This chapter includes the following information:

- PeopleSoft Samples
- Code Example
- Building Message Definitions

PeopleSoft Samples

The PeopleSoft Samples provided show the general flow of a connector program.

NOTE	You must build the message definition, described in "Building Message Definitions", before running samples on either Windows NT/2000 or Solaris. Ensure that iPlanet Web Server and iPlanet Application Server are up and running.

Activation

The PeopleSoft sample consist of servlets which activate PeopleSoft programs that access the Peoplesoft management system.

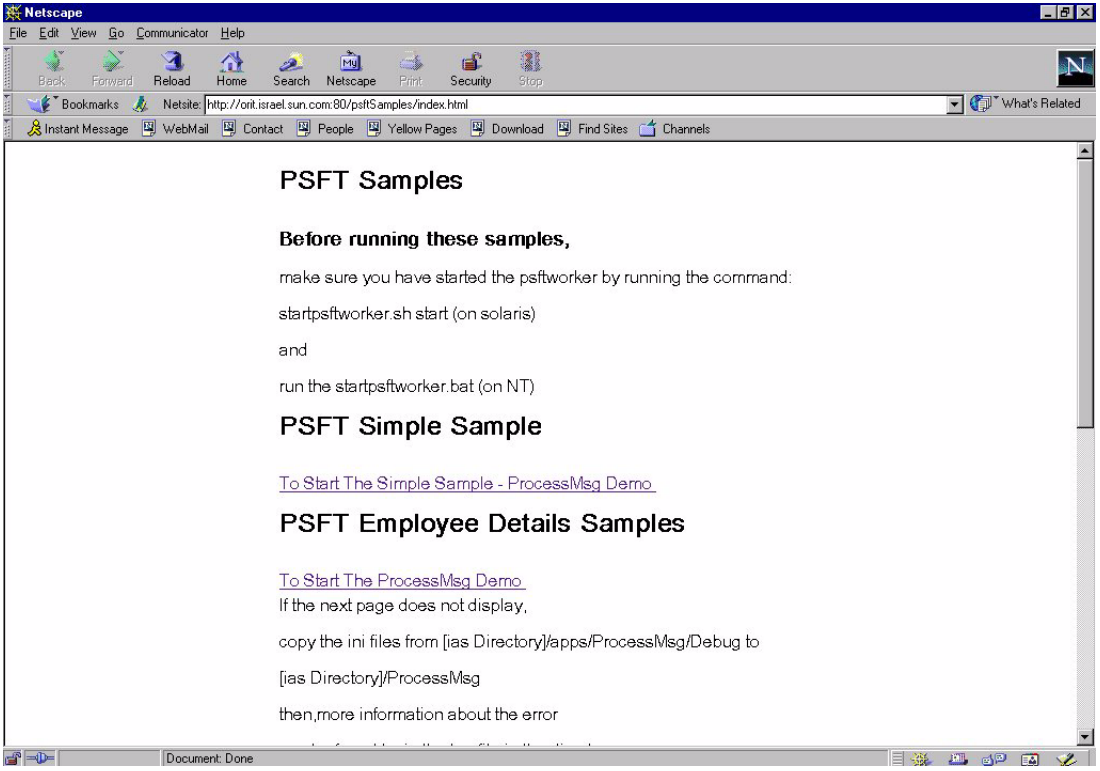
You must have set up the environment before activating the samples. See Post Installation Issues in Chapter 2 of the *iPlanet Application Server Enterprise Connector for PeopleSoft Administrator's Guide*.

To Run the Peoplesoft Samples on Windows NT/2000

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

   The PeopleSoft sample script is displayed with links to the samples.
3. Click on the link "To Start The Simple Sample - ProcessMsg Demo" to activate the sample shown in Figure 4-1.

Figure 4-1 PeopleSoft Customer Details Samples



To run the Employee Details Sample on Solaris

1. From <iPlanet Install directory>/ias/APPS/bin, type:

   ```
   ./startpsftworker.sh start
   ```

2. Start your browser.

3. Enter the following URL: <host name>:<web server port>/psftSamples.

   The PSFT Employee Details Samples appears, showing links to the ProcessMsg and SearchDialog samples.

4. Click on: `To Start The ProcessMsg Demo` link

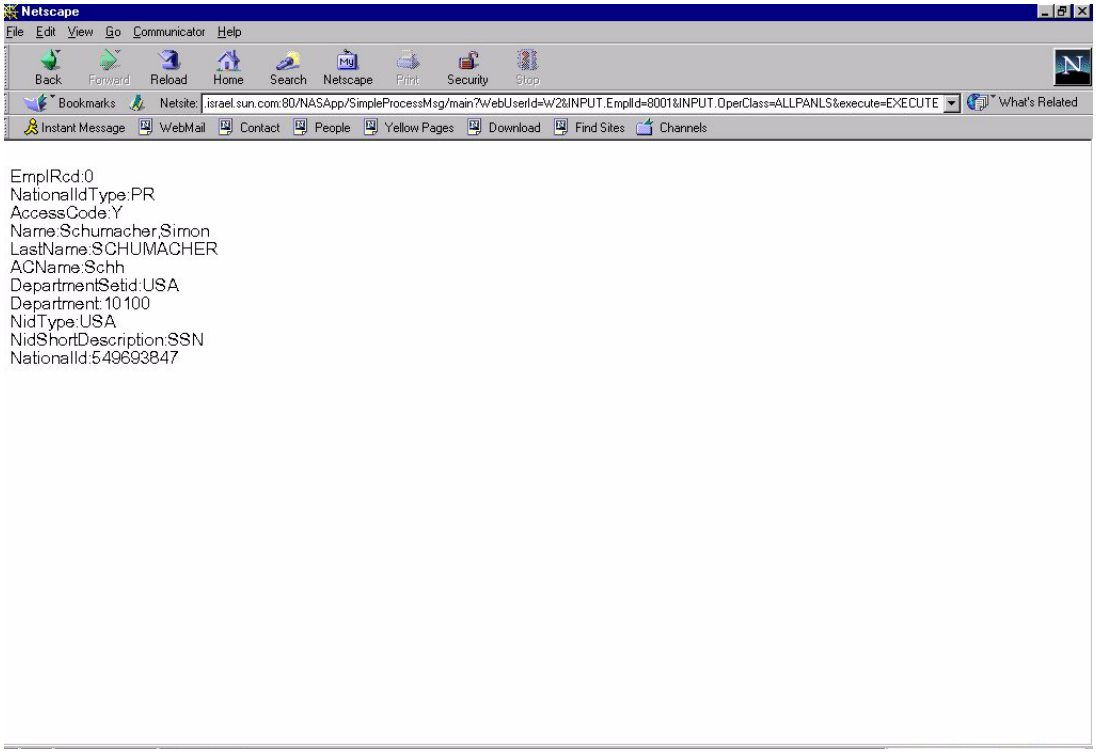
   The Employee Details samples opening screen appears.

5. Type in the information in the text fields, or accept the default values, and click on Execute.

Figure 4-2 startForm.jsp

6. After the data finishes processing the ProcessMsg End Form appears.

Figure 4-3 Employee's Details.



Code Example

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

The code listings for the Process Message and Search Dialog samples are detailed in Code Example 4-1 and Code Example 4-2.

Code Example 4-1 SimpleProcessMsg.java

```
package SimpleProcessMsg;

import java.util.*;
import javax.servlet.http.*;
import java.io.*;
```

Code Example 4-1 SimpleProcessMsg.java

```
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;
    }

    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("PSFT", "connection");
        else
            return null;
    }
```

Code Example 4-1 SimpleProcessMsg.java

```
    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("PSFT",
"UpdateEmployeeDetails.DisplayEmployeeDetails.ProcessMsg");
                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(request,data,out);
                   prop = fn.getProperties();
                   if( prop != null )
                      setInputProperties(prop);
                   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");
```

Code Example 4-1 SimpleProcessMsg.java

```
            } // 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
   {

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

data.setAttrFString("INPUT.OperClass",getInputString(request,"INPUT.OperClass",
out));
   }

   public void getOutputData(ServletOutputStream out,IBSPDataObject data)
throws BspException,IOException
   {
      out.println("\n<BR>EmplRcd:"+data.getAttrInt("OUTPUT.EmplRcd"));
```

Code Example 4-1 SimpleProcessMsg.java

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

out.println("\n<BR>AccessCode:"+data.getAttrFString("OUTPUT.AccessCode"));
        out.println("\n<BR>Name:"+ data.getAttrFString("OUTPUT.Name"));
        out.println("\n<BR>LastName:"+ data.getAttrFString("OUTPUT.LastName"));
        out.println("\n<BR>ACName:"+ data.getAttrFString("OUTPUT.ACName"));
        out.println("\n<BR>DepartmentSetid:"+
data.getAttrFString("OUTPUT.DepartmentSetid"));
        out.println("\n<BR>Department:"+
data.getAttrFString("OUTPUT.Department"));
        out.println("\n<BR>NidType:"+ data.getAttrFString("OUTPUT.NidType"));
        out.println("\n<BR>NidShortDescription:"+
data.getAttrFString("OUTPUT.NidShortDescription"));
        out.println("\n<BR>NationalId:"+
data.getAttrFString("OUTPUT.NationalId"));

    }
    public void setInputProperties(IBSPDataObject prop)    throws BspException
    {
        prop.setAttrInt("ProcessingMessages.OperationType",0);
    }


}//main
```

Search Dialog Demo Code Examples

Code Example 4-2 SimpleProcessMsg.jsp

```
<HTML>
<HEAD>
     <TITLE>SimpleProcessMsg Demo</TITLE>
</HEAD>
<BODY>
<center>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600" >
     <tr>
     <td VALIGN=TOP WIDTH="420" >
     <form action="/NASApp/SimpleProcessMsg/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>
              SimpleProcessMsg 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">EmplId:</font></td>
                 <td><font face="arial, helvetica, sans-serif">
                    <input type="text" name="INPUT.EmplId"  size=20 maxsize=50
value="8001" ></font>
                 </td>
              </tr>
              <tr>
                 <td><font face="arial, helvetica, sans-serif" color=white
size="-1">OperClass:</font></td>
                 <td><font face="arial, helvetica, sans-serif">
                    <input type="text" name="INPUT.OperClass"  size=20
maxsize=50 value="ALLPANLS" ></font>
                 </td>
              </tr>
           </TABLE>
           </center>
           </td>
        </tr>
```

Code Example 4-2 SimpleProcessMsg.jsp

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

Building Message Definitions

Message Definitions are the metadata, which describe the format of data associated with a particular PeopleSoft panel. This section includes the procedures for building your own message definitions. You must build the message definition before you run a sample.

For more information on PeopleSoft application development, go to the PeopleSoft web site at:

`http://www.peoplesoft.com/en/us/products/technology/index.html`

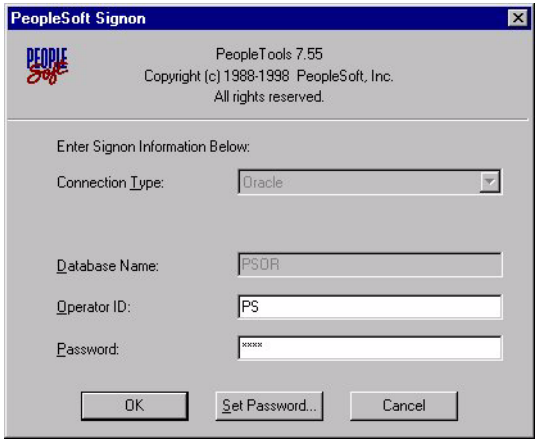
You build message definitions by using the following procedures:

- To Create a Project
- To Select Project Options
- To Define a New Employee Record Field
- To Add Existing Fields To Your Project
- To Define The Record And Build A Search Dialog Box
- To Set the Record Field Properties
- To Set the Edit Field Options
- To Set Table Space
- To Build The Record
- To Build a Panel
- To Define a Panel Group For This Panel
- To Define Your Menu
- To Define a New Business Process
- To Define a Message Agent
- To Specify the Message Agent Field Map
- To Give Authorization To The Panel
- To Initiate a Step

To Create a Project

1. Select Start > Programs > PeopleSoft 7.5 > Application Designer.

   The PeopleSoft Signon screen appears.



2. Select Oracle from the drop-down list in the Connection Type list box.

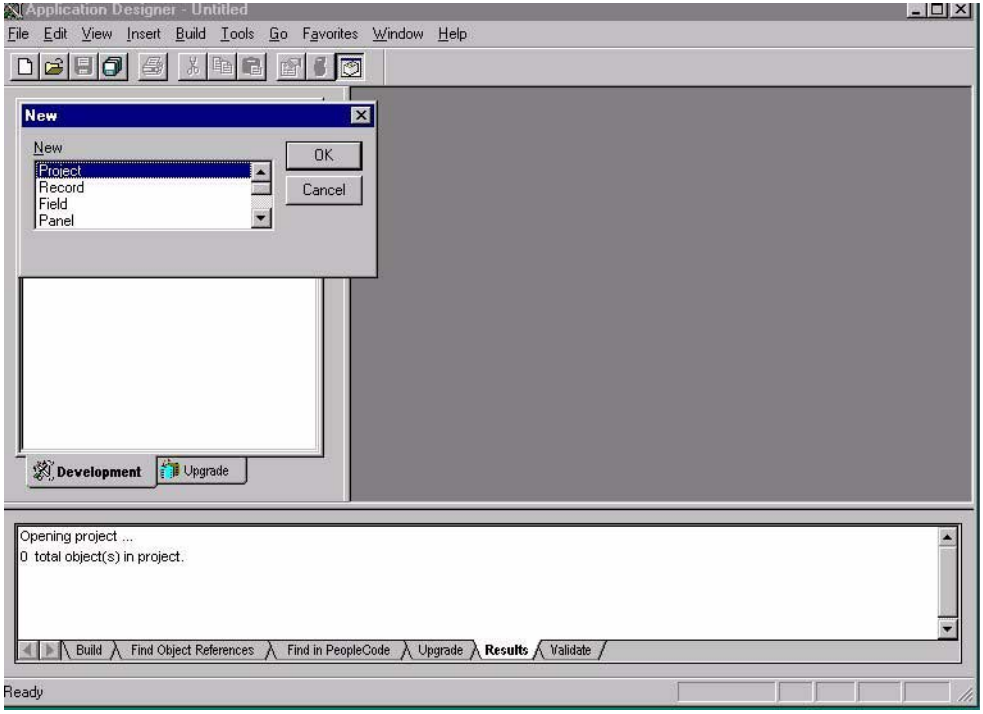
   Accept the default PSOR in the Database Name dialog box.

3. Type in your Operator ID and Password and click on the OK button.

   The PeopleSoft Application Designer window appears.

4. Select File > New.

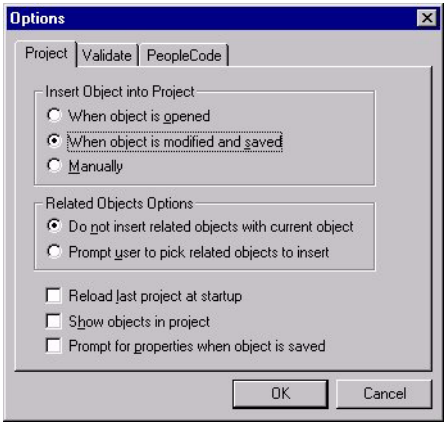
5. From the New dialog, select Project and click OK.



To Select Project Options

1. Select Tool > Options.

   The Options dialog box appears.



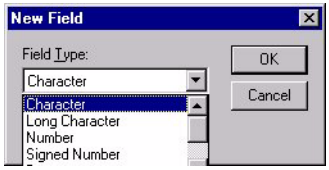
2. Select *When Object is modified and saved* from the *Insert Object into Project* radio button group.
3. Select D*o not insert related objects with current object* radio button from the *Related Objects Options* radio button group.
4. Click OK to save these options.

To Define a New Employee Record Field

You must include the new field, EMPL_RCD, to run the samples.

1. Select File > New > Field.
2. Click OK.

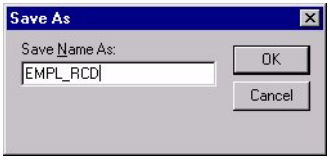
   The New Field drop-down list box appears.



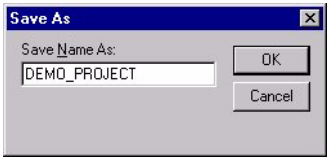
3. Select Number from the drop-down list and click OK.
4. Type:
   - Employee Record in the *Long Name* field.
   - EmplRcd in the *Short Name* field.

5. Dismiss the dialog box.

6. Select File > Save As and type EMPL_RCD in the *Save Name As* field of the dialog box.



7. Click OK to save the new field.
8. Select File > Save Project As.
9. Type DEMO_PROJECT in the *Save Name As* text box and then click OK.

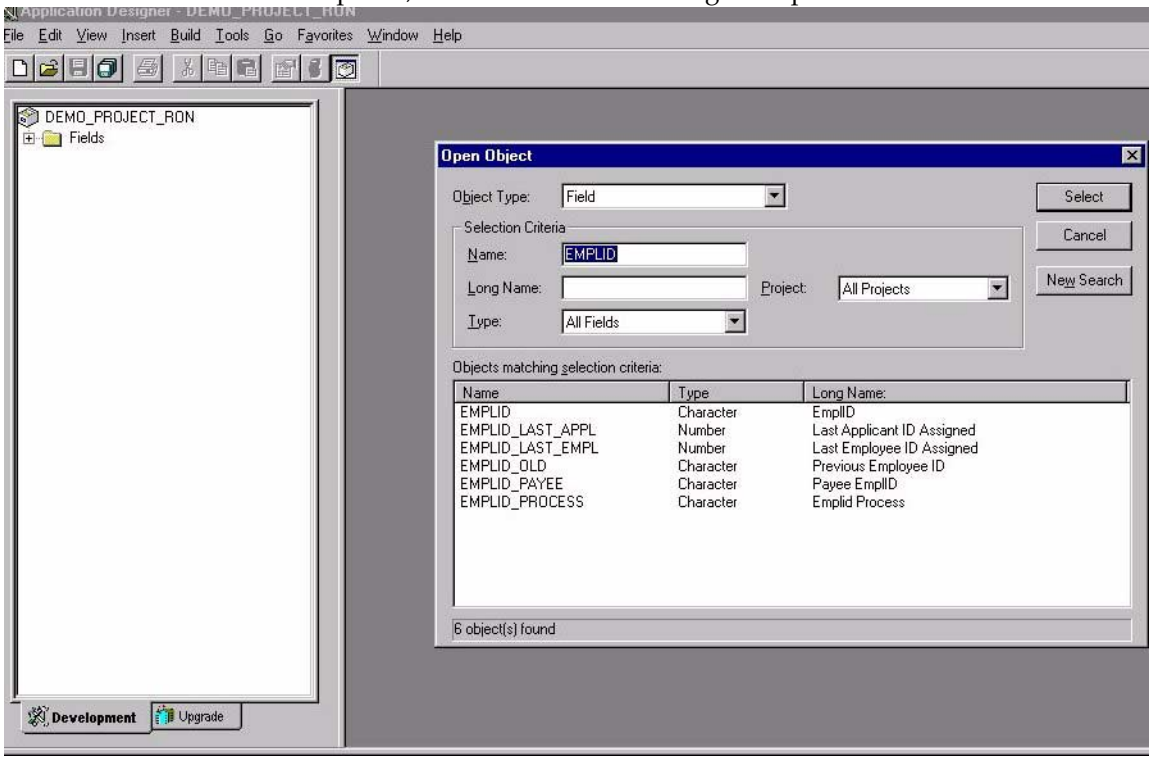


To Add Existing Fields To Your Project

1. Select File > Open.
2. Select Field from the Object Type drop-down list box.
3. Type EMPLID into the *Name* text field.

4. Click on Select.

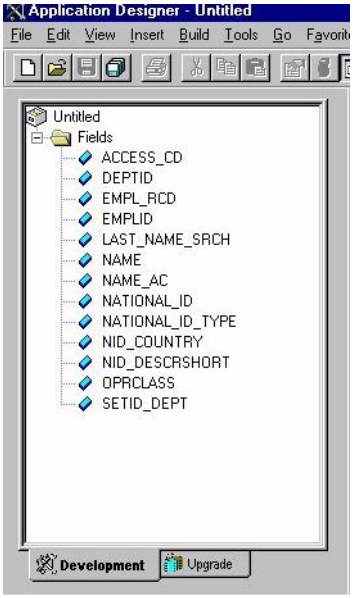
   A list of all fields that start with that prefix appear. If there is only one field with the prefix, the field definition dialog box opens.



5. Double-click the EMPLID field to open the field definition dialog box.
6. Press <F7> to insert the currently selected object into the project.
7. Close the Open Object dialog box.
8. Repeat the procedure starting with step 1 to add the following fields to the project.
   - EMPL_RCD
   - OPRCLASS
   - ACCESS_CD
   - NAME
   - LAST_NAME_SRCH

- SETID_DEPT
- DEPTID
- NAME_AC
- NID_COUNTRY
- NATIONAL_ID_TYPE
- NID_DESCRSHORT
- NATIONAL_ID

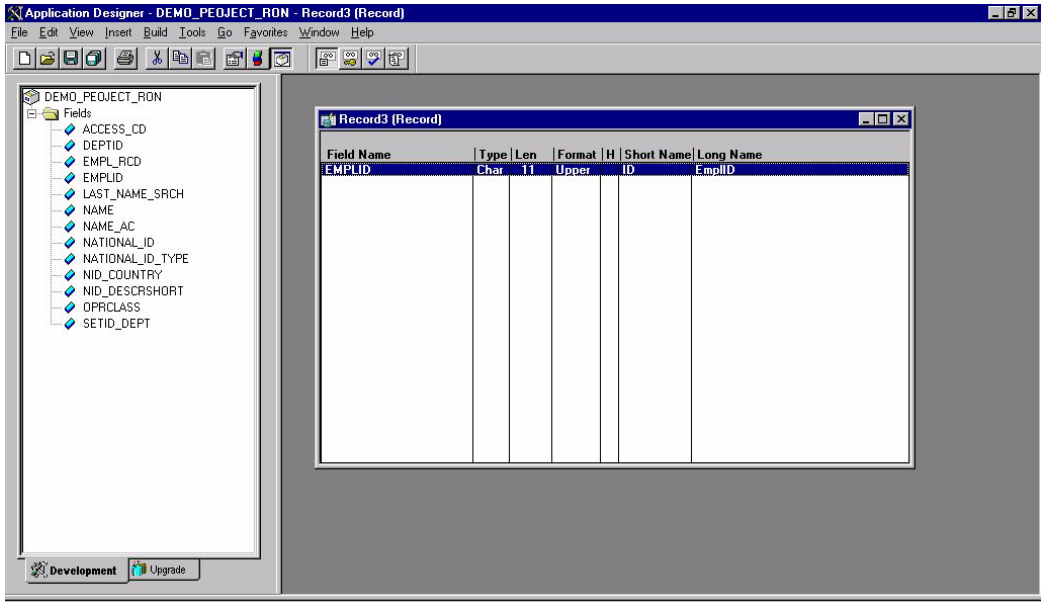
Your screen should look like this when you finish inserting all the necessary fields.



To Define The Record And Build A Search Dialog Box

1. Select File > New.
2. Select Record from the Object Type drop-down list box.
3. Click OK.

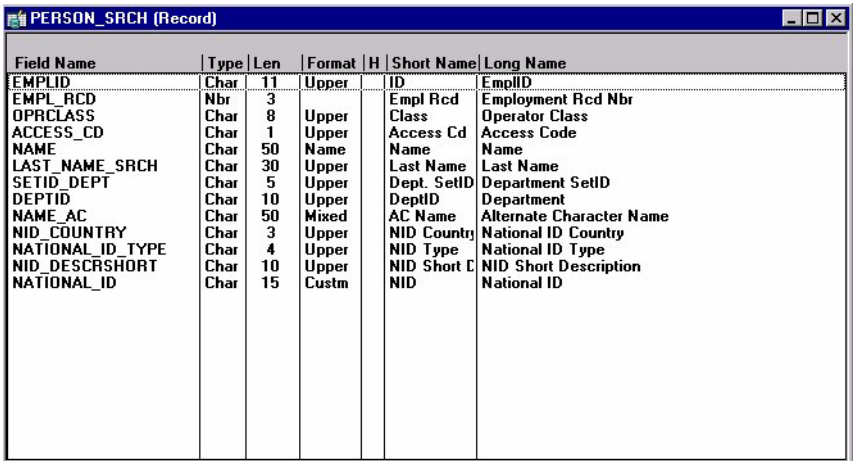
4. Drag-and-drop the EMPLID field to the new record.



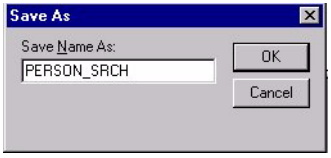
NOTE The sequence of inserting fields in the record is important.

5. Drag-and-drop all the other designated fields to the record in the order shown in the following figure.

   When you are finished your record should look like this.



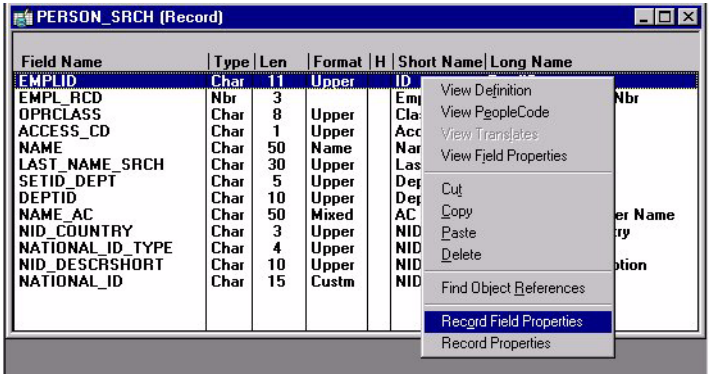
6. Select File > Save As.
7. Type in the new record name and click OK.



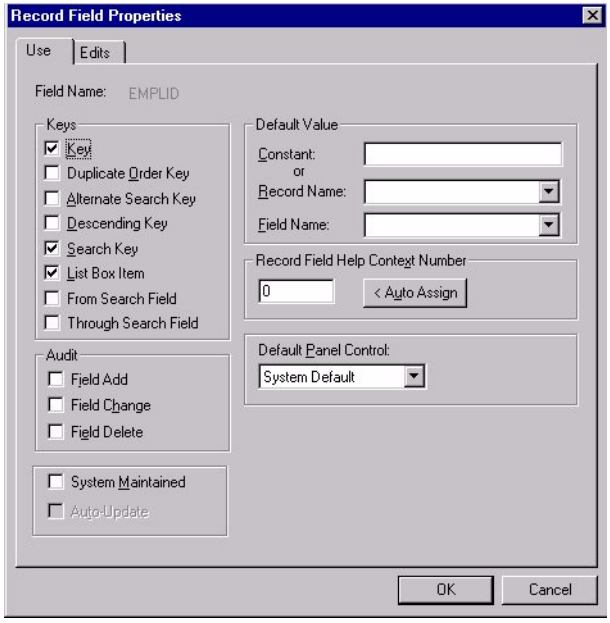
To Set the Record Field Properties

NOTE In order to be able to perform search operations you must select the Key, Alternative Key, and List Box Item for each relevant field in the new record.

1. Select the EMPLID field and then click the right mouse button to display the list box.



2. Choose Record Field Properties to display the Record Field properties dialog box.



3. Mark the following check boxes:
   - Key
   - Search Key
   - List Box Item (this is automatically selected if you select either Key or Search Key)
4. Click OK.
5. Continue selecting each field in turn until the record field properties have been specified for each field according to the following table.

Table 4-1 Message Definition Options

Field	Key	Search	List	System
EMPLID	Key	Yes	Yes	No
EMPL_RCD		No	Yes	No
OPRCLASS	Key	No	No	No
ACCESS_CD		No	No	No
NAME	Alt	No	Yes	No
LAST_NAME_SEARCH	Alt	No	Yes	No
SETID_DEPT*	Alt	No	Yes	No
DEPTID*	Alt	No	Yes	No
NAME_AC	Alt	No	Yes	No
NID_COUNTRY		No	No	No
NATIONAL_ID_TYPE		No	No	No
NID_DESCRSHORT		No	No	No
NATIONAL_ID		No	No	No

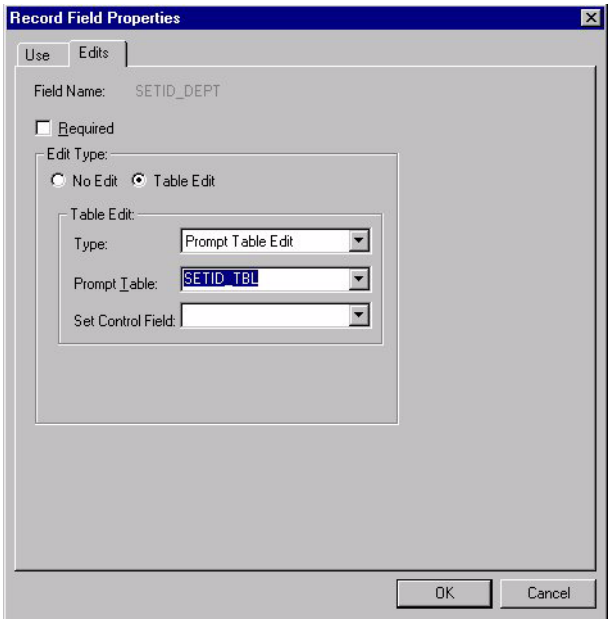
NOTE Alt - check the Alternate Search Key check box

To Set the Edit Field Options

NOTE * These edit options must be set only for the SETID_DEPT and DEPTID fields.

1. Select the SEPTID_DEPT field and then click the right mouse button to display the list box.
2. Choose Record Field Properties to display the Record Field properties dialog box.

3. Select the Edit tab to display the Record Field properties Edit dialog box.



4. Select the Edit options as specified in Table 4-2.

Table 4-2 Record Field Properties Edit Options

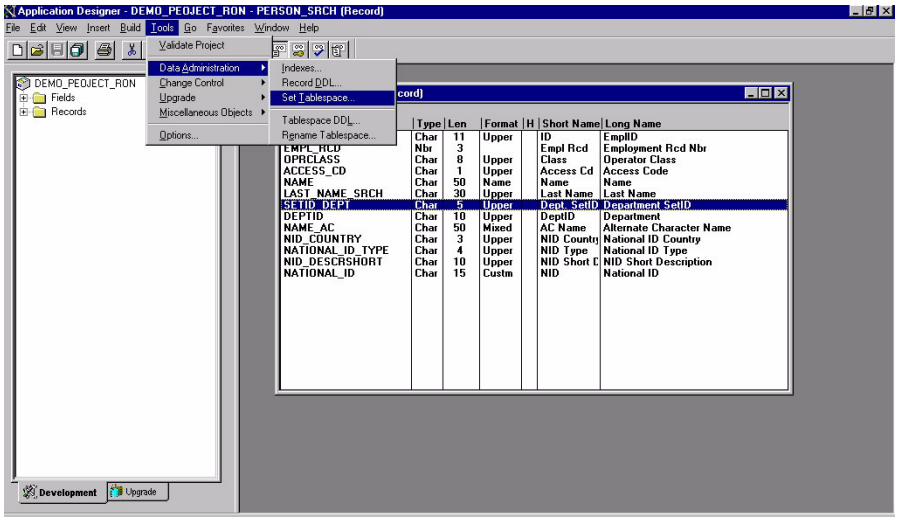
Record	Edit Type	Type	Prompt_Table
SETID_TBL	Table Edit	Prompt Table Edit	SETID_TBL
PERS_DEPT_VW	Table Edit	Prompt Table Edit	PERS_DEPT_VW

5. Select File > Save.

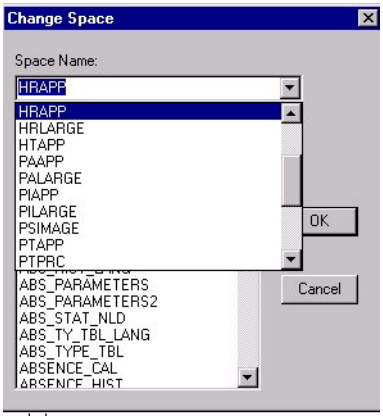
To Set Table Space

1. Select Tools > DataAdministration > Set_TableSpace.

2. Click on Set_TableSpace to display the Select Tools data administrator.



3. Click on Set_TableSpace to display the Change Space dialog box.

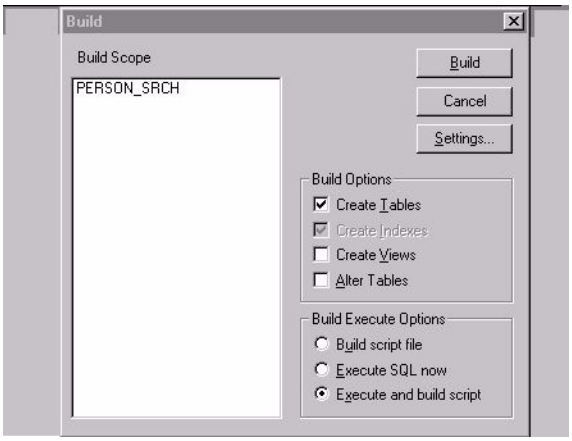


4. Select HRAPP and click OK button.
5. Click Save All button.

To Build The Record

1. Select the record you built.

2. Select Build > Current Object to display the Build dialog box.



3. Select:

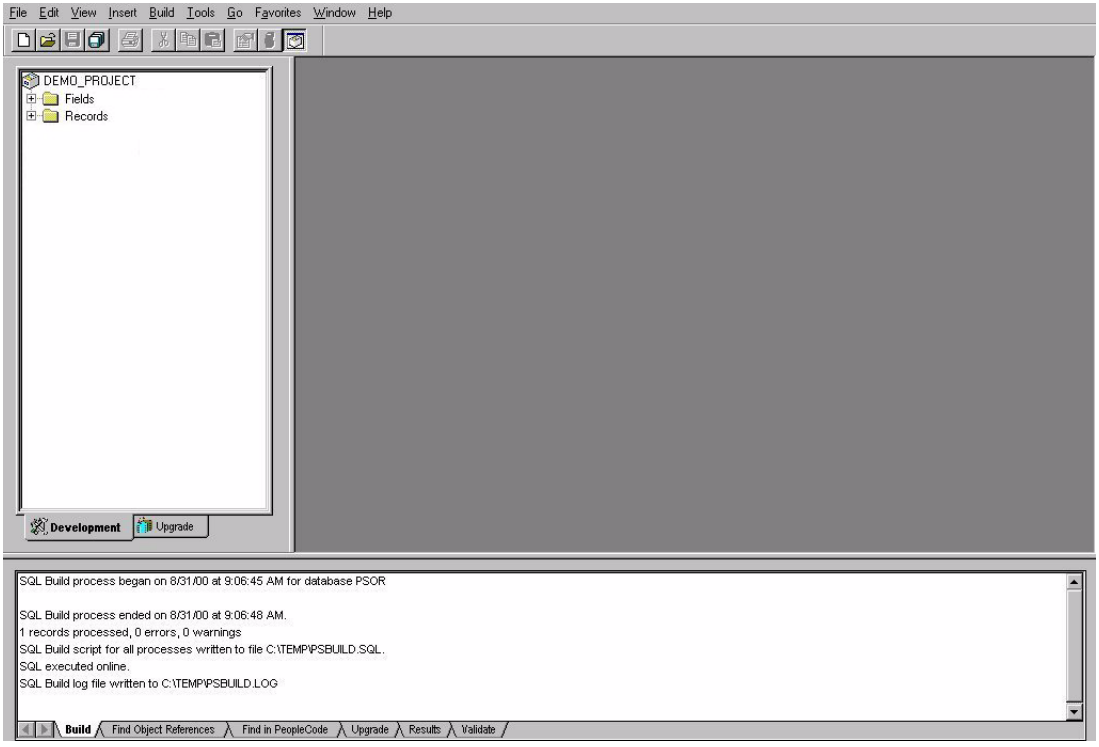
   - Create Tables check box

     The Create Indexes check box is automatically selected also.

   - Execute-and-build-script button.

4. To build the new record, click the Build button.

   Check that there are no error messages in the Output window.



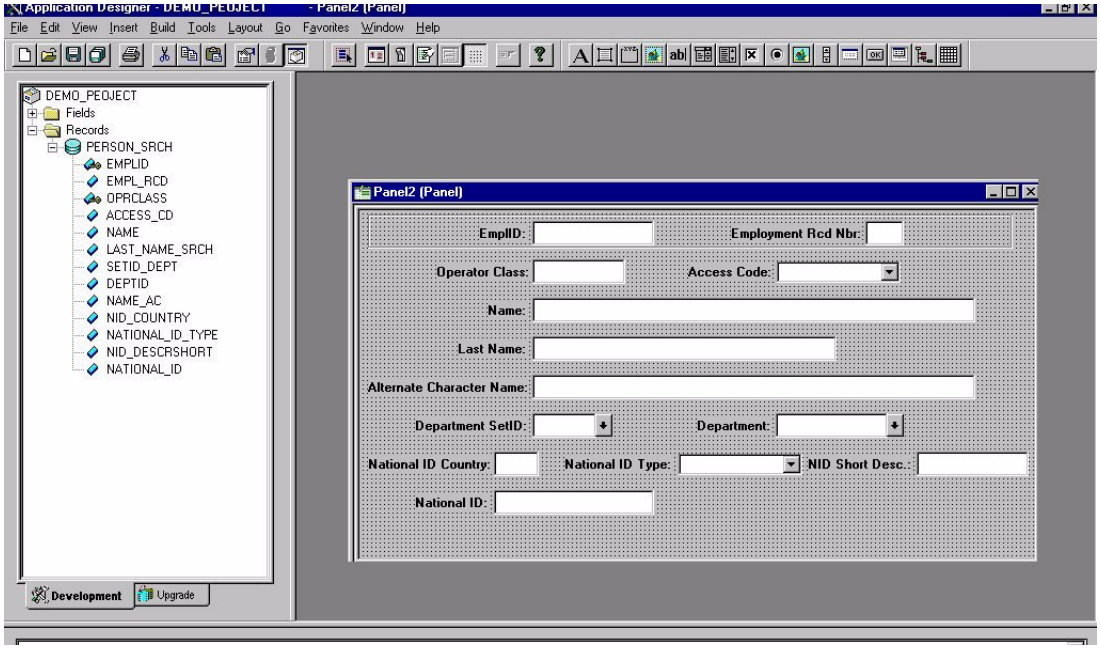
   If there are error messages, consult your PeopleSoft documentation.

To Build a Panel

1. Select File > New.
2. Select Panel from the Object Type drop-down list box.
3. Click OK.

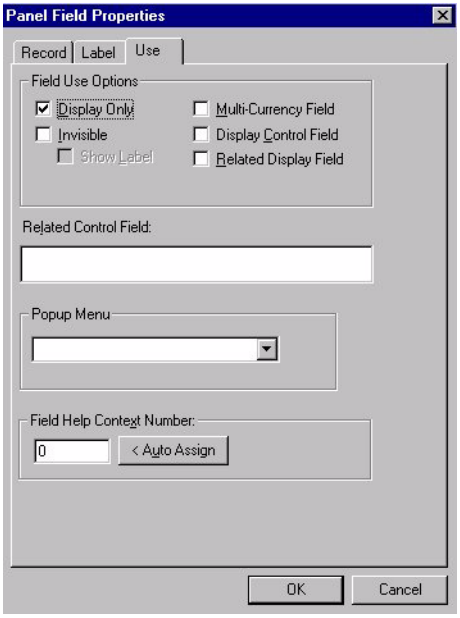
4. Drag-and-drop the fields from the new record that you built onto the panel.

   The field prompt display sequence will be the same as the sequence that the fields were dragged-and-dropped.



5. Double-click the EMPLID field, the Panel Field Properties dialog box is displayed.

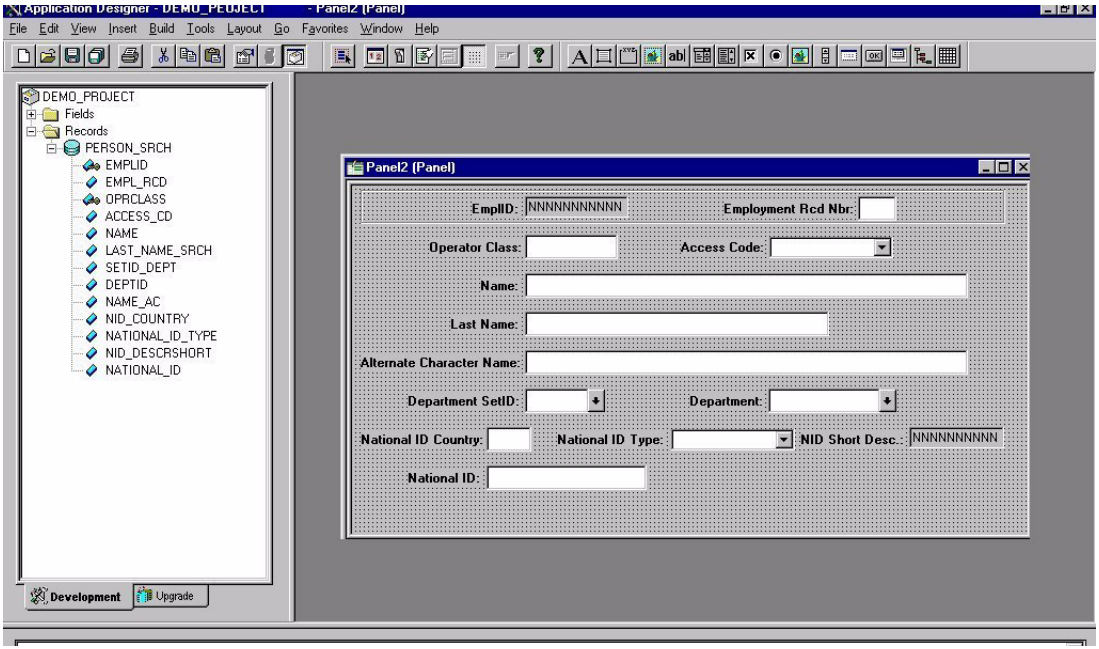
6. Click the Use tab to display the Field Use Options dialog box.



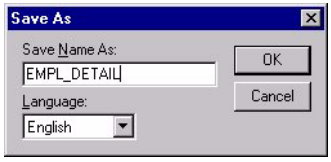
   Mark the Display Only check box.

7. Click OK.
8. Double-click the National ID Type field to display the Panel Field Properties dialog box.
9. Mark the Related Display Field check box.
10. Click OK.
11. Double-click the NID Short Desc. field to display the Panel Field Properties dialog box.
12. Mark the Display Control Field check box and then select National ID Type Field from the Related Control Field list.
13. Click OK.

14. Select File > Save As.



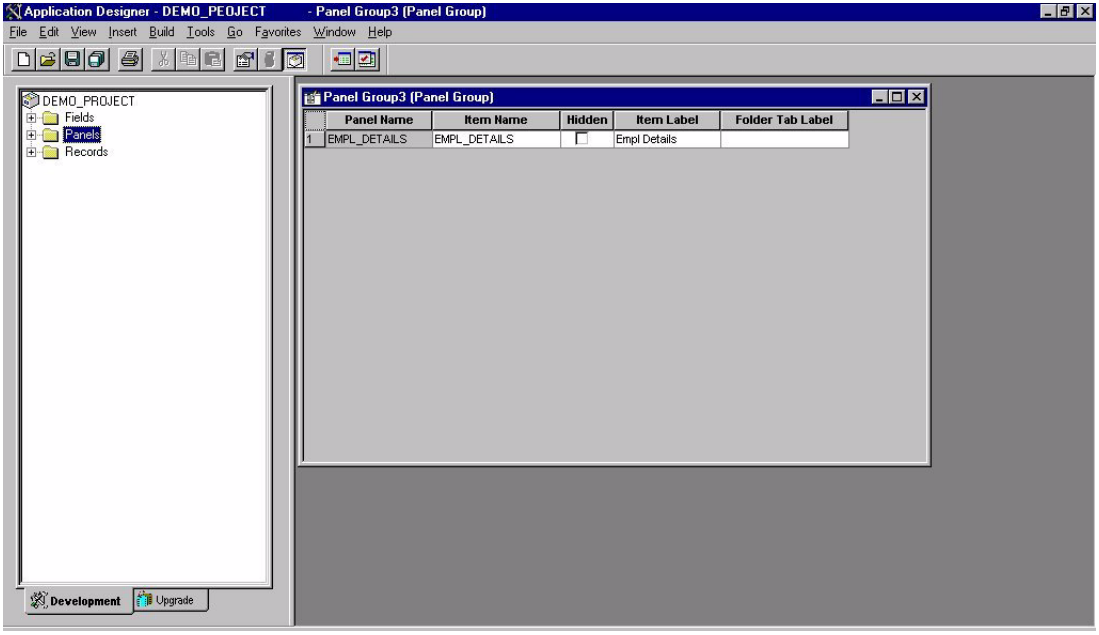
15. In the Save As dialog box, type in the Panel name and then click OK.



To Define a Panel Group For This Panel

1. Select File>New.
2. Select Panel Group from the Object Type drop-down list box.
3. Click OK.

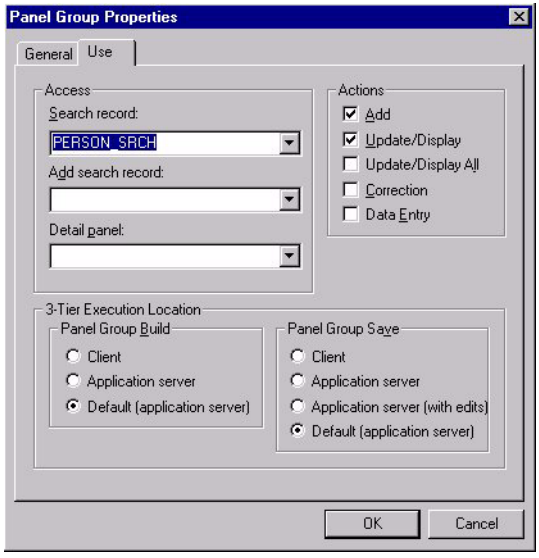
4. Drag-and-drop the EMPL_DETAILS panel to the Panel Group window.



5. Click the Property button.

The Panel Group Properties dialog box is displayed.

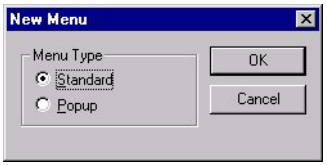
6. Click on the Use tab.



7. From the Search record drop-down menu, select the Record you defined in the previous step.
8. Accept the default Actions: Add and Update/Display, and the default 3-Tier Execution Location.
9. Click OK.
10. Select File > Save As.
11. In the Save As dialog box, type the Panel Group name,
12. Select the default Market type from the drop-down list
13. Click then OK button.

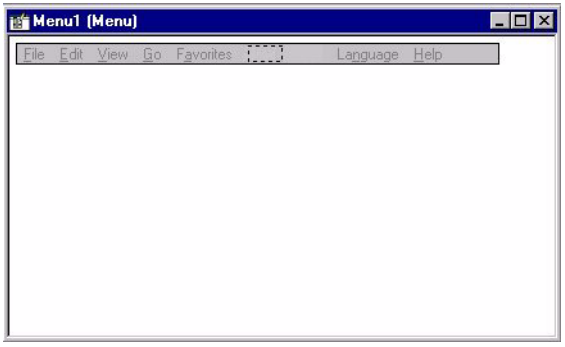
To Define Your Menu

1. Select File > New.
2. Select Menu from the Object Type drop-down list box and then click OK.

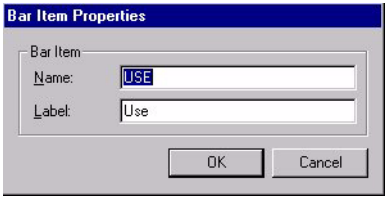


The Menu bar is displayed.

3. Double-click on the broken-line box on the menu bar.



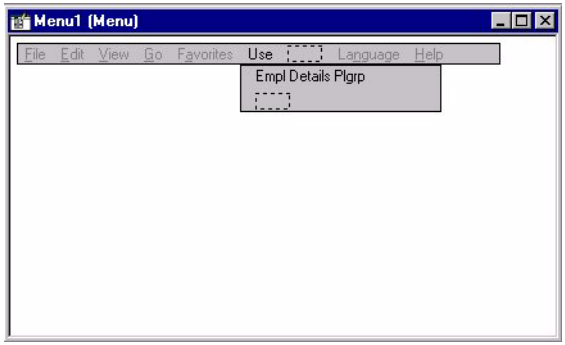
The Bar Item Properties dialog box is displayed.



4. Type *Use* in the Name and Label text boxes and then click OK.

The Menu bar now displays a new Bar item: Use.

5. Drag -and drop the panel group to the menu item.

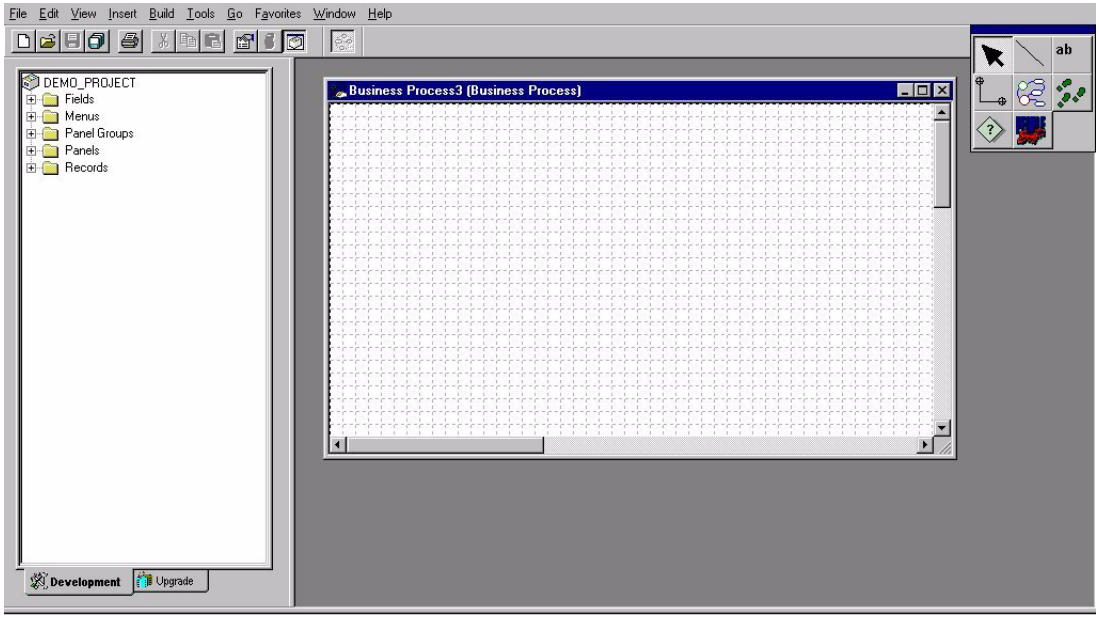


6. Select File > Save As.
7. Type the Menu name in the Save As dialog box, and click OK.

To Define a New Business Process

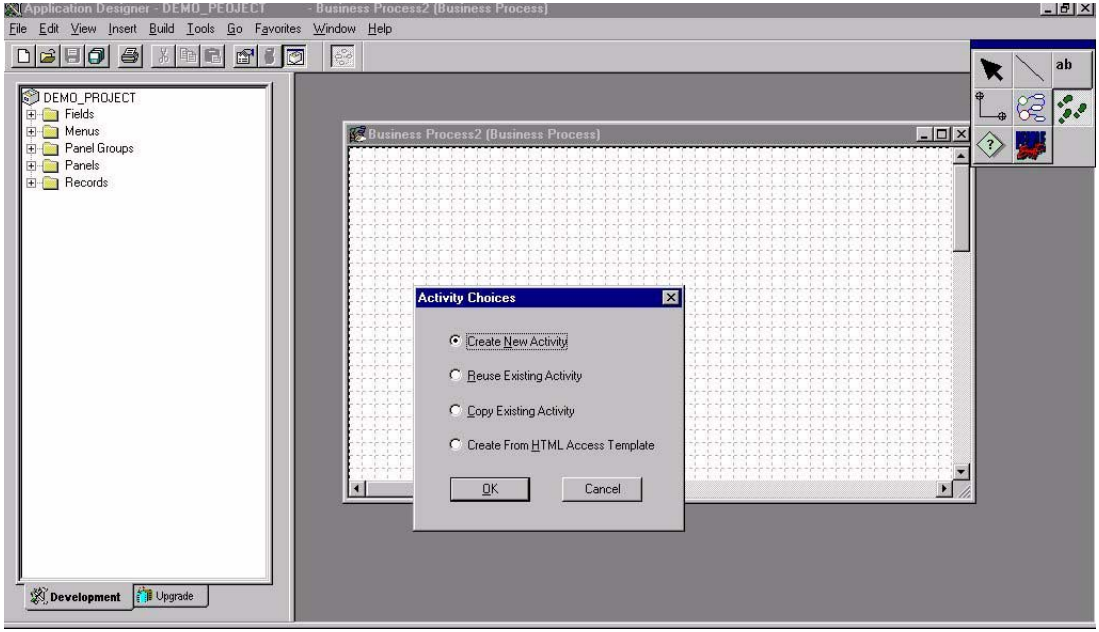
1. Select File > New > Business Process and click OK.

   The Business Process workspace and toolbar appears.



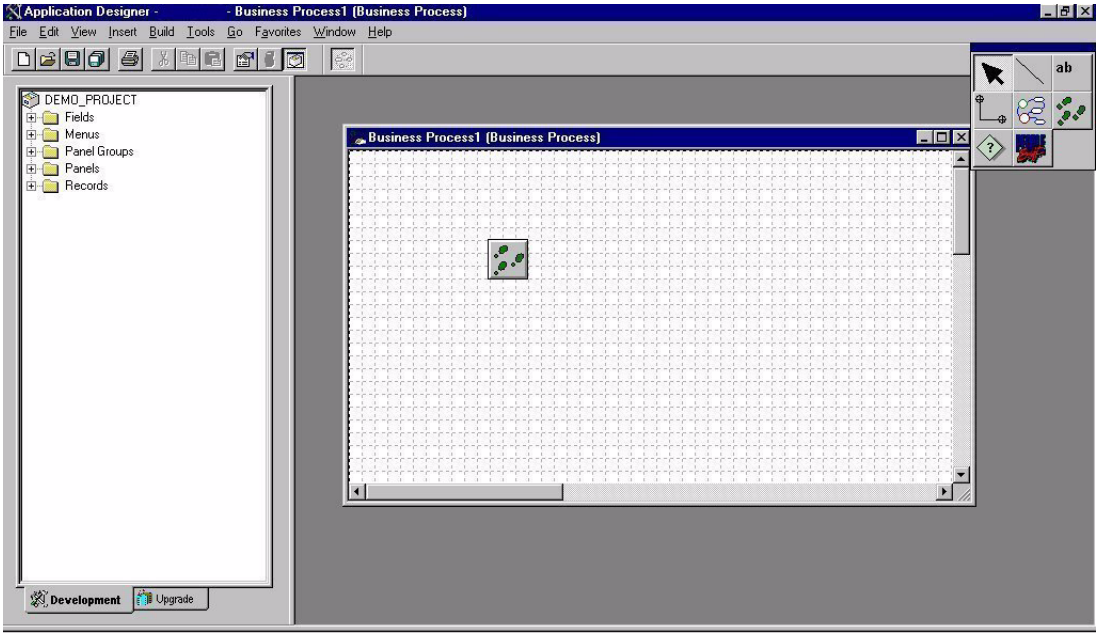
2. Select the Activity icon by clicking with the left-mouse button.

3. When the Activity is at the location you want, click the left-mouse button to display the Activity Choice dialog box.



4. Select the Create New Activity radio button and click OK.

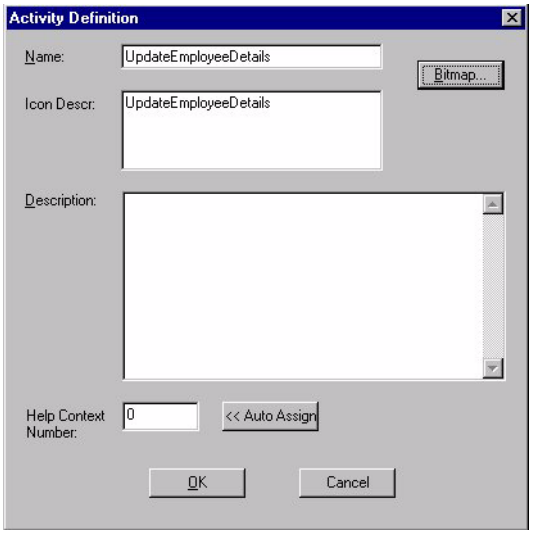
   An Activity tool appears on the worksheet.



5. Select the Activity tool and then click the right-mouse button.

   The Activity Definition dialog box appears.

6. Type the activity definition in the Name text box and then press the tab key on you keyboard; the name is automatically copied to the Icon Description. text box.

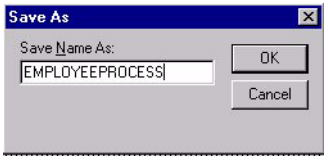


Do not leave spaces in the activity definition name.

7. Click OK.

8. Select File > Save.

Type the name of the business process into the text box of the Save As dialog box.



9. Click OK to create a business process.

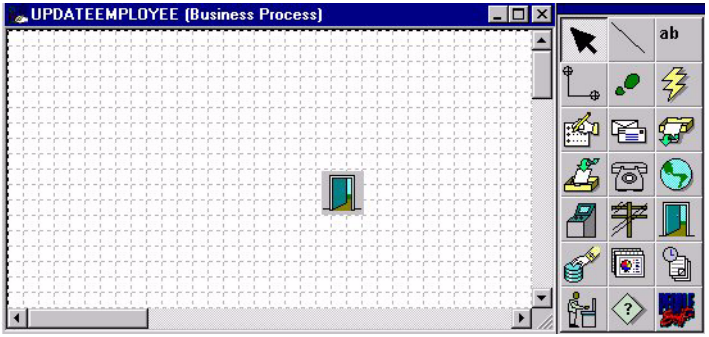
To Define a Message Agent

1. Double-click the activity.

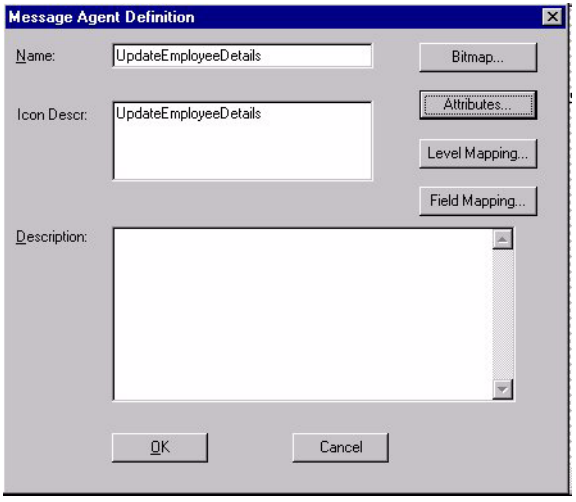
A business process palette appears.

2. Left click on the Other App icon.

3. Position the cursor in the worksheet.
4. Click the left-mouse button



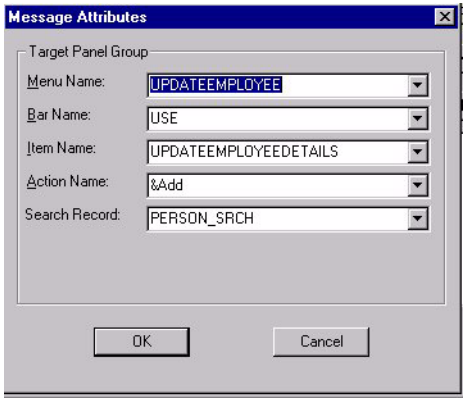
5. Click the right-mouse button to display the Message Agent Definition dialog box.



6. Type in the Message Agent Definition name.

   Do not leave spaces within the Message Agent Definition name.

7. Click the Attribute button and select the fields from the Message Attribute dialog box drop-down lists.

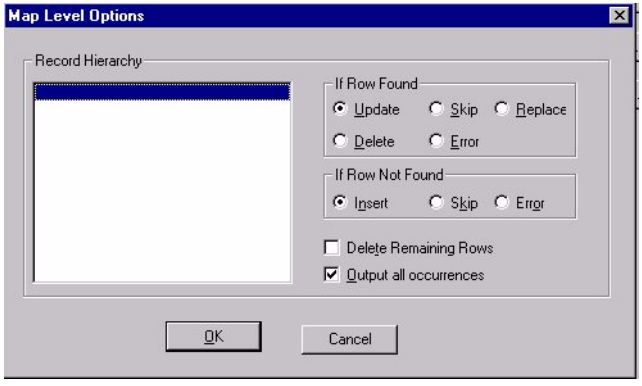


8. Click OK.

   You need to build a new message for each action.

9. Click the Level Mapping button and select the following radio buttons and checkbox

   - If Row Found - Update
   - If Row Not found - Insert
   - Output all occurrences - Mark check box.

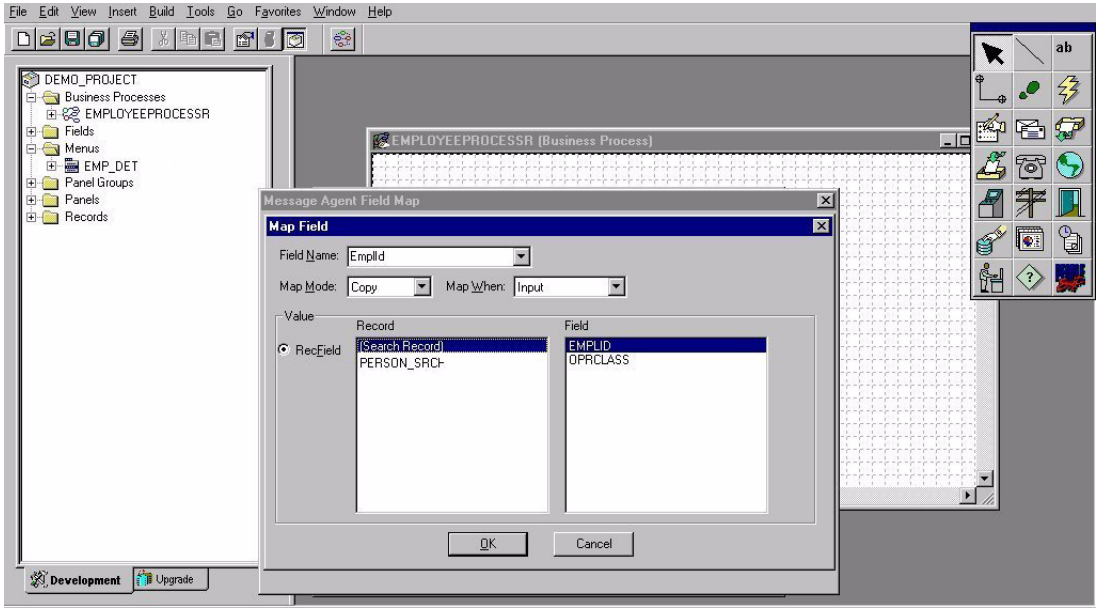


10. Click OK.

To Specify the Message Agent Field Map

1. Click on the Field Mapping button on the Message Agent Definition dialog box.
2. Click the Add button.

   For each field you must select the source Record, the Field name, and Input and/or Output.
3. Type in the Field name.



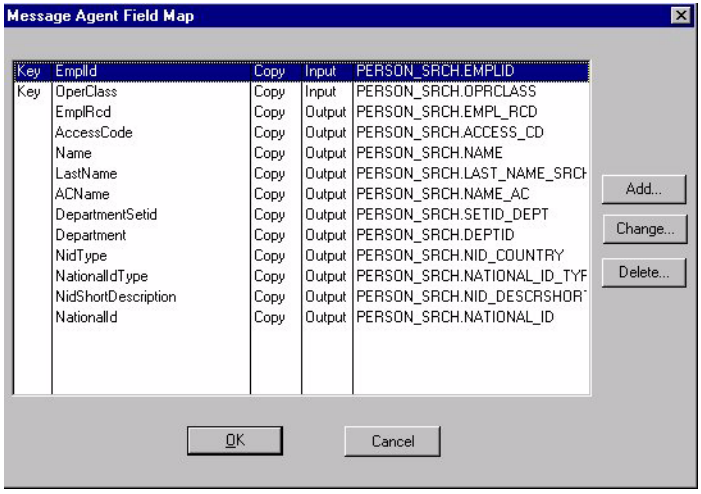
4. Select the record from the Record list box.

   EmplId and OperClass must be selected from the Search Record.

   All other records must be selected from the appropriate source record.

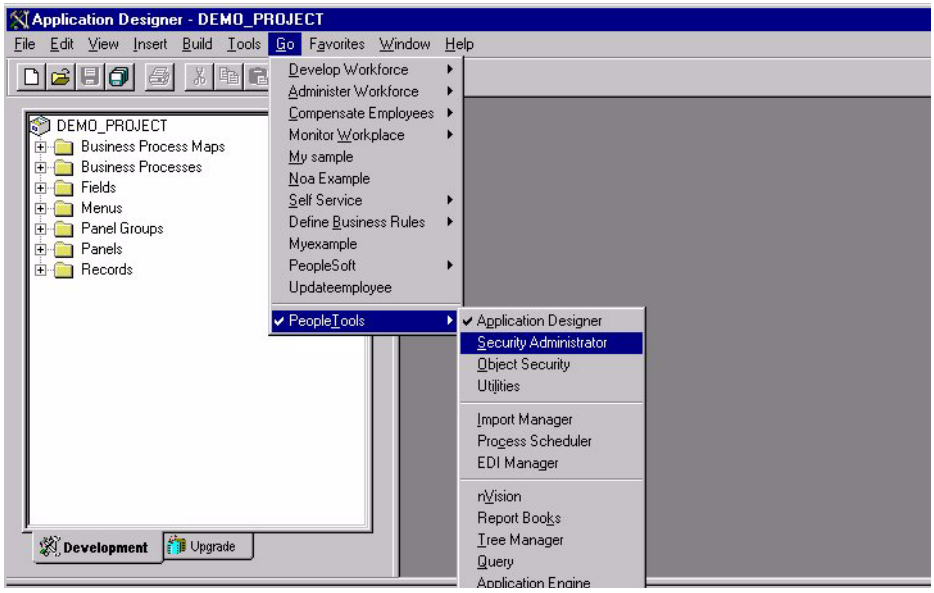
NOTE When you define a Message Definition that is used to add a record, the key field must have in the Map When drop-down list box Input.

5. The following table is the end result of the complete process.

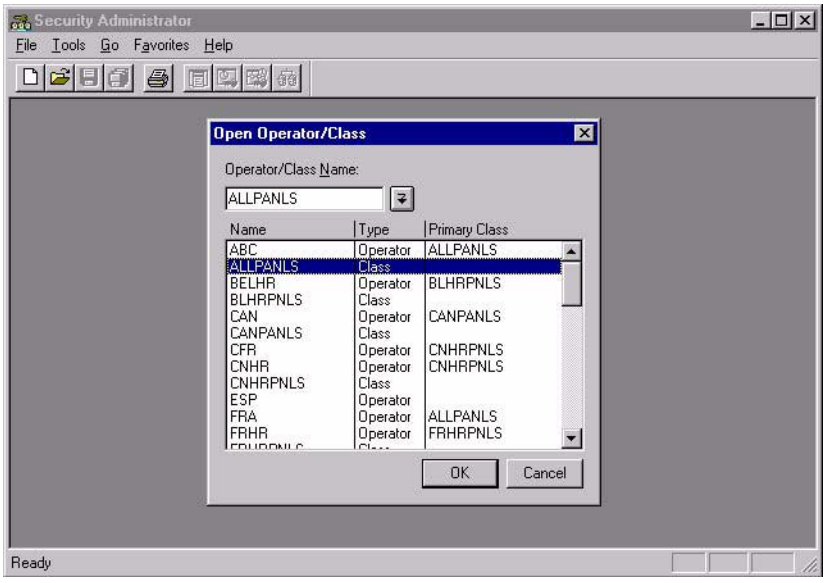


To Give Authorization To The Panel

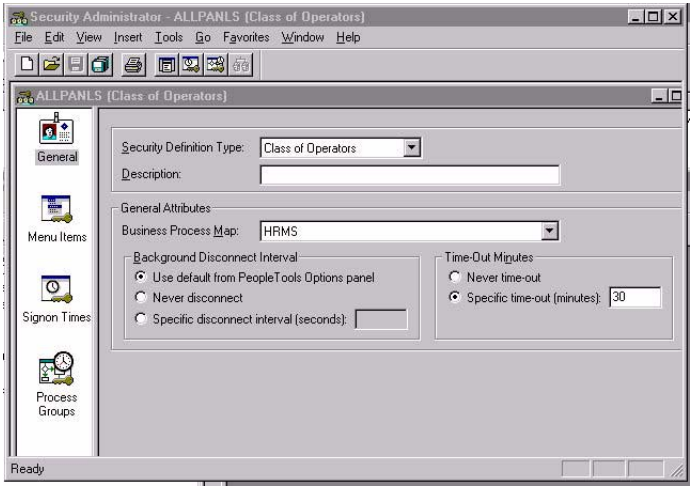
1. Select Go > PeopleTools > Security Administrator.



2. Select File > Open.

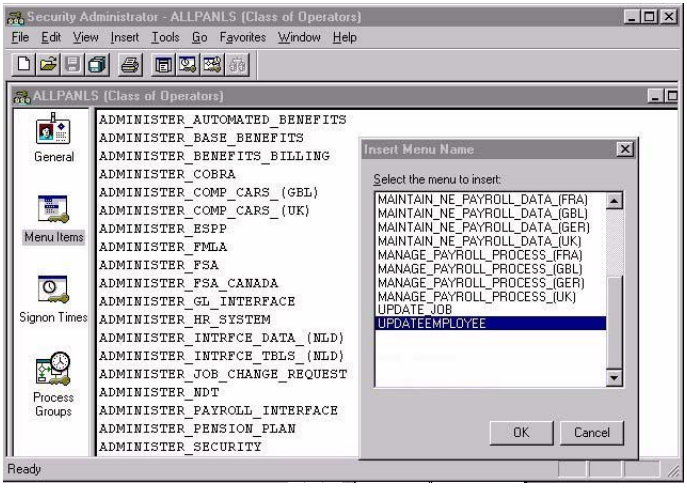


3. Select ALLPANLS and click OK.

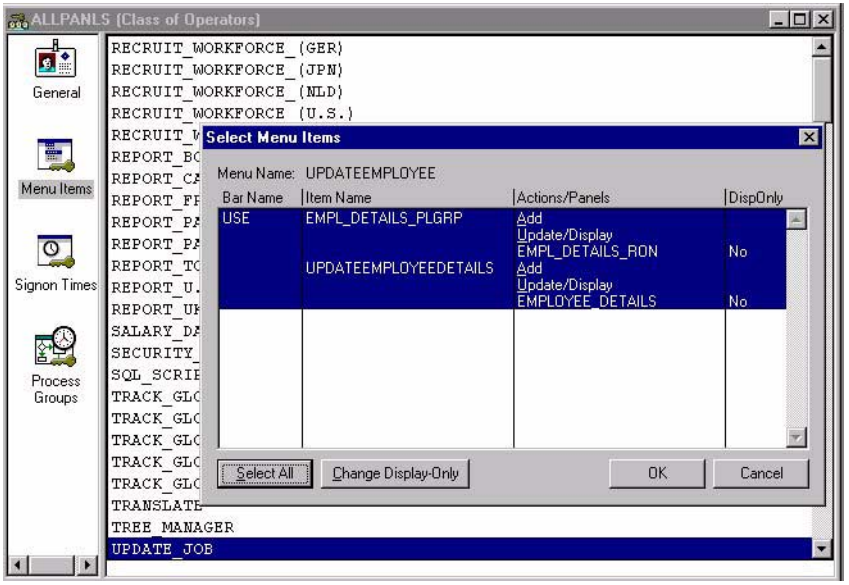


4. Click the Insert-Menu-Name button on the tool bar to display the Insert Name panel.

The Select Menu Items panel appears.



5. Select the Menu you created.

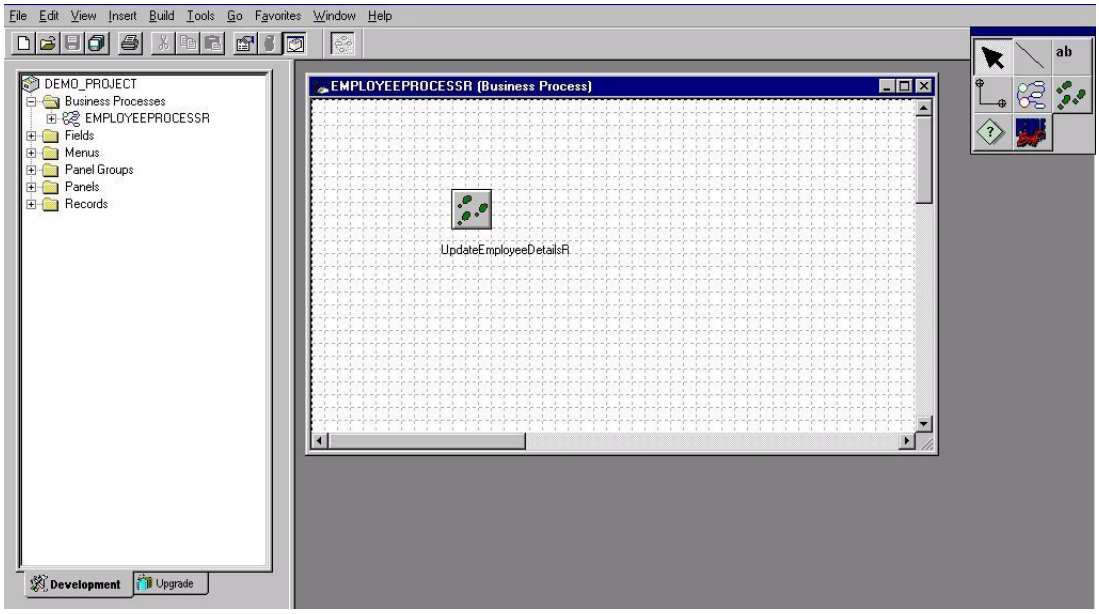


6. Click the Select All button.
7. Click OK.
8. Click the Save All button.

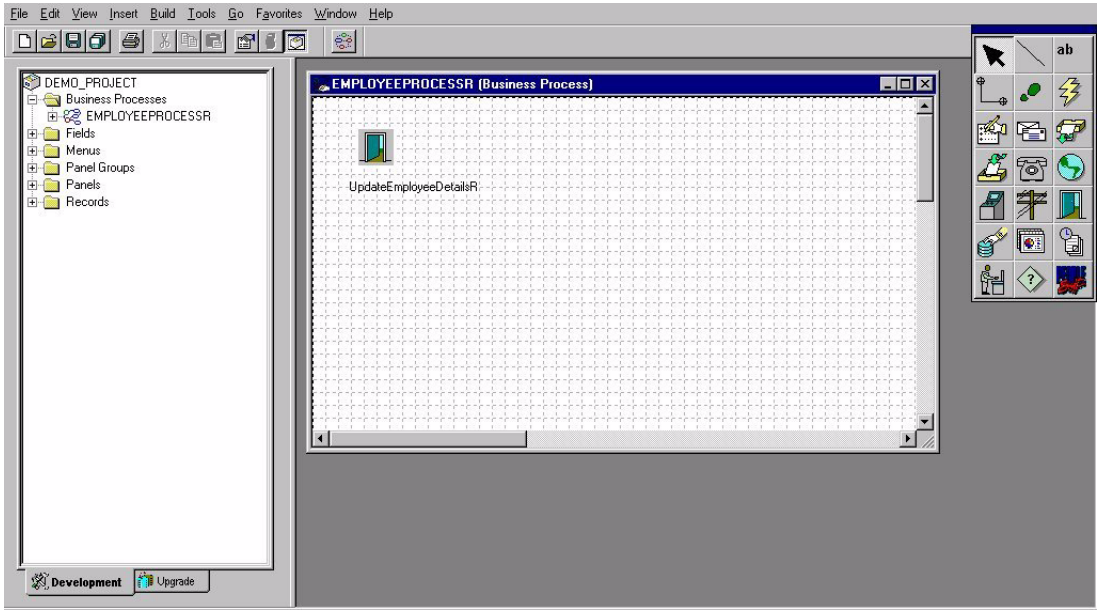
To Initiate a Step

1. Open the Business Process folder.

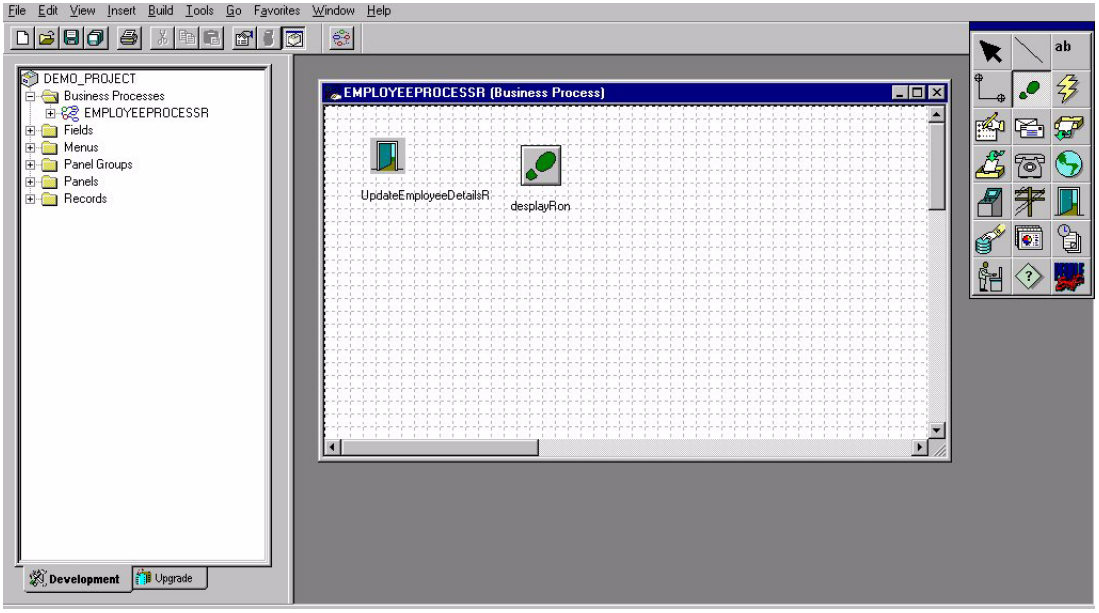
2. Double-click the Business Process that you created.



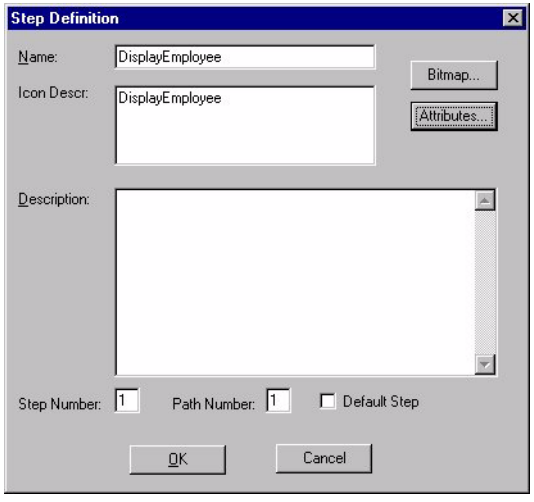
3. Double-click on the Business Process to display the Activities.



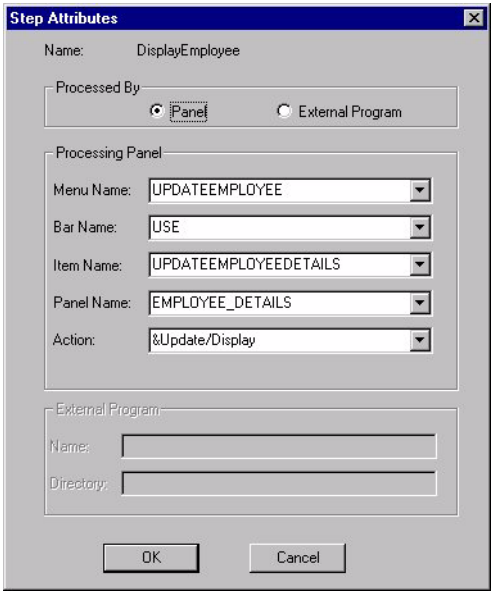
4. Click on the Step icon on the Business Process palette.
5. Position the cursor on the worksheet and click the left-mouse button.



6. Click the right-mouse button to open the Step Definition panel.
7. Type in the Name and Icon Description.



8. Click the Attribute button.



9. Select the inputs from the drop-down lists.
10. Click OK.
11. Choose the Save As button.

NOTE You need to build a new step for each action.

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 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 Message Agent. text.
3	Output data is too long, maximum 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.
7	Type not supported in Data Object {object name}, {field name}, type {field type}.

Table A-1 Error Messages

Error Message Code *msgid*	Error Message
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	Output Retrieve error. Failed in {operation}, {explanation}. {explanation}.
17	Output Retrieve error. Failed in {operation}. Rows count is 0. {explanation}. {explanation}.
18	Output Retrieve error. Failed in {operation}. Fields count is 0. {explanation}. {explanation}
19	Output Retrieve error. Failed in {operation}, No matching rows. {explanation}. {explanation}.
20	Output Retrieve error. Failed in {operation}. Field {name} not found. {explanation}. {explanation}.
21	Output Retrieve error. Failed to get Field information. Field {name} not found.
22	Invalid MappedEntity object.
23	Failed to get field poolname from pSPConfig object.
24	Pool {name} is not defined.
25	Psft is not a dynamic pool.

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 PeopleSoft 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")
;
                }
}
```

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.

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. 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.

iPlanet Application Server The iPlanet Application Server 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.

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