## Contents

**Preface**

<table>
<thead>
<tr>
<th>PeopleSoft Component Interfaces Preface</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeopleSoft Component Interfaces</td>
<td>ix</td>
</tr>
</tbody>
</table>

**Chapter 1**

**Getting Started with PeopleSoft Component Interfaces**

<table>
<thead>
<tr>
<th>Overview</th>
<th>1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Implementing PeopleSoft Component Interfaces</th>
<th>1</th>
</tr>
</thead>
</table>

| Implementing the Excel to Component Interfaces Utility | 2 |

**Chapter 2**

**Understanding Component Interfaces**

<table>
<thead>
<tr>
<th>Understanding Component Interfaces</th>
<th>3</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Component Interface Architecture</th>
<th>3</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Component Interface Attributes</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Keys</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Collections</th>
<th>5</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>5</th>
</tr>
</thead>
</table>

| Component Interface Definitions and Views | 5 |

**Chapter 3**

**Developing Component Interfaces**

<table>
<thead>
<tr>
<th>Creating Component Interface Definitions</th>
<th>9</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Understanding Creating Component Interface Definitions</th>
<th>9</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Creating New Component Interfaces</th>
<th>10</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Naming Component Interface Definitions</th>
<th>12</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Associating Component Interfaces with Menus</th>
<th>12</th>
</tr>
</thead>
</table>

| Determining the Fields to Expose in Component Interfaces | 14 |
Chapter 4

Programming Component Interfaces Using PeopleCode ................................................................. 59

Understanding PeopleCode Behavior and Limitations ................................................................. 59
  PeopleCode Event and Function Behavior .................................................................................. 59
CopyRowset Language Considerations ......................................................................................... 61
  Limitations of Client-Only PeopleCode .................................................................................... 61
Generating PeopleCode Templates to Access Component Interfaces .............................................. 61
Understanding PeopleCode Templates .......................................................................................... 63

Using Keys ................................................................................................................................. 14
  Understanding Keys .................................................................................................................. 14
  Adding and Deleting Keys ......................................................................................................... 16
Setting Properties ....................................................................................................................... 16
  Understanding Standard Properties ......................................................................................... 16
  Creating User-Defined Properties ......................................................................................... 29
  Deleting User-Defined Properties ......................................................................................... 30
  Renaming User-Defined Properties ....................................................................................... 30
  Creating Reference Properties ............................................................................................... 31
  Making Properties Read-Only .................................................................................................. 33
Working with Collections ........................................................................................................... 33
Working with Methods ................................................................................................................. 34
  Understanding Session Functions and Methods .................................................................. 34
  Understanding Standard Methods ......................................................................................... 35
  Understanding Collection Methods ....................................................................................... 39
  Enabling and Disabling Standard Methods ......................................................................... 43
  Creating User-Defined Methods ........................................................................................... 44
  Exporting User-Defined Methods .......................................................................................... 46
Validating Component Interfaces ................................................................................................. 46
Setting Security Options ................................................................................................................ 47
Testing Component Interfaces ....................................................................................................... 48
  Searching Component Interfaces to Test ................................................................................ 48
  Testing Component Interfaces .................................................................................................. 51
  Determining ItemByKey Parameters ...................................................................................... 54
Understanding Synchronization ..................................................................................................... 55
Writing Component Interface Programs ....................................................................................... 56
Understanding Runtime Considerations ....................................................................................... 57
  General Considerations ............................................................................................................ 57
  Scope Conflicts .......................................................................................................................... 57
  Interactive Mode ....................................................................................................................... 58
Chapter 5

Programming Component Interfaces in Java ................................................................. 65
  Building APIs in Java .............................................................................................. 65
  Setting Up the Java Environment ............................................................................ 66
  Generating Java Runtime Code Templates .............................................................. 66
  Understanding the Java Template ............................................................................ 68

Chapter 6

Programming Component Interfaces in C++ .............................................................. 71
  Building APIs for C++ ............................................................................................ 71
  Setting Up the C++ Environment ............................................................................ 72
    Setting Up Client Machines to Access C++ APIs .................................................. 72
    Configuring Compilers for C++ Projects .............................................................. 72
  Generating C++ Runtime Code Templates ............................................................. 73
  Understanding the C++ Template .......................................................................... 75

Chapter 7

Programming Component Interfaces in COM ............................................................ 79
  Building APIs for COM .......................................................................................... 79
  Setting Up the COM Environment .......................................................................... 80
  Generating Visual Basic Runtime Code Templates ................................................. 81
  Understanding Visual Basic Templates .................................................................. 82

Chapter 8

Using the Component Interface Software Development Kit ....................................... 87
  Understanding the Component Interface SDK ....................................................... 87
    Component Interface SDK Samples ...................................................................... 87
  Prerequisites for Using the Component Interface SDK ......................................... 88
  Using the SDK_BUS_EXPENSES Test Page ............................................................ 88
  Testing the SDK_BUS_EXP Component Interface ................................................ 89
  Using the Component Interface SDK Sample in Java and C++ ............................ 89
    Understanding using the Component Interface SDK Samples in Java and C++ .... 89
    Building the Component Interface SDK Sample (Java) ..................................... 89
    Building the Component Interface Sample (C++) .............................................. 89
Appendix A

Creating Component Interface-Based Services ......................................................... 133

Understanding Generating Component Interfaced-Based Services .......................... 133

Index ............................................................................................................................. 135
PeopleSoft Component Interfaces Preface

This book describes PeopleSoft component interfaces. It is written for programmers who will be accessing PeopleSoft components, usually using external systems.

---

PeopleSoft Component Interfaces

A PeopleSoft component interface is a PeopleTools definition that you create in PeopleSoft Application Designer. It enables synchronous access to a PeopleSoft component from another application.

---

PeopleBooks and the Online PeopleSoft Library

A companion PeopleBook called PeopleBooks and the Online PeopleSoft Library contains general information, including:

- Understanding the PeopleSoft online library and related documentation.
- How to send PeopleSoft documentation comments and suggestions to Oracle.
- How to access hosted PeopleBooks, downloadable HTML PeopleBooks, and downloadable PDF PeopleBooks as well as documentation updates.
- Understanding PeopleBook structure.
- Typographical conventions and visual cues used in PeopleBooks.
- ISO country codes and currency codes.
- PeopleBooks that are common across multiple applications.
- Common elements used in PeopleBooks.
- Navigating the PeopleBooks interface and searching the PeopleSoft online library.
- Displaying and printing screen shots and graphics in PeopleBooks.
- How to manage the PeopleSoft online library including full-text searching and configuring a reverse proxy server.
- Understanding documentation integration and how to integrate customized documentation into the library.
- Glossary of useful PeopleSoft terms that are used in PeopleBooks.

You can find this companion PeopleBook in your PeopleSoft online library.
Chapter 1

Getting Started with PeopleSoft Component Interfaces

This chapter provides an overview of PeopleSoft component interfaces and discusses how to implement them.

Overview

A component interface is a set of application programming interfaces (APIs) that you can use to access and modify PeopleSoft database information programmatically. PeopleSoft component interfaces expose a PeopleSoft component (a set of pages grouped for a business purpose) for synchronous access from another application (PeopleCode, Java, C/C++, or Component Object Model [COM]). A PeopleCode program or an external program (Java, C/C++, or COM) can view, enter, manipulate, and access PeopleSoft component data, business logic, and functionality. Additionally, you can use the Component Interface Tester to check the validity of your component interface and the Excel to Component Interface Utility to manage your data.

Component interfaces are created in PeopleSoft Application Designer, so you should ensure that you are familiar with PeopleTools and Application Designer.


This section provides information to consider before you begin to use PeopleSoft component interfaces. In addition to implementation considerations presented in this section, take advantage of all PeopleSoft sources of information, including the installation guides, release notes, and PeopleBooks.

Implementing PeopleSoft Component Interfaces

PeopleSoft PeopleTools include the functionality to create component interfaces for your applications.

Complete the following tasks before you begin creating component interfaces for your implementation:

- Install your Application according to the installation guide for your database type.
  
  See the PeopleSoft installation guide for your platform and product line.

- Establish a user profile that gives you access to PeopleSoft Application Designer and any other processes that you will use.
  
  See Enterprise PeopleTools 8.50 PeopleBook: Security Administration, "Understanding PeopleSoft Security."
Implementing the Excel to Component Interfaces Utility

PeopleSoft provides the Excel to Component Interface utility that enables you to upload data from Microsoft Excel into your PeopleSoft database. Several tasks are involved in setting up the Excel to Component Interfaces Utility.

Chapter 2

Understanding Component Interfaces

This chapter provides an overview of component interfaces and discusses:

- Component interface architecture.
- Component interface attributes.
- Component interface definitions and views.

Understanding Component Interfaces

A component interface enables exposure of a PeopleSoft component (a set of pages grouped together for a business purpose) for synchronous access from another application (such as PeopleCode, Java, C/C++, COM, or XML). Component interfaces can be viewed as "black boxes" that encapsulate PeopleSoft data and business processes, and hide the details of the underlying page and data. Component interfaces can be used to integrate one application with another application or with external systems. Component interfaces execute the business logic built into the component and as a result, they provide a higher level of data validation than a simple SQL insert.

A component interface maps to one, and only one, PeopleSoft component. However, you can create multiple component interfaces for the same component. You create component interfaces in PeopleSoft Application Designer. Record fields on the component are mapped to the keys and properties of the component interface. Methods are used to find, create, modify, or delete data.

Component Interface Architecture

The component interface architecture comprises three fundamental elements—components, component interfaces, and the component interface API.

Every component interface has the following main attributes:

- Name.
- Keys (Get keys, Create keys, and Find keys).
- Properties and collections (fields and records).
- Methods.
Note. In most cases, component interfaces act like their associated components, meaning that PeopleCode events typically trigger in the same order as the component. However, several runtime exceptions relate to component interfaces and PeopleCode processing and search dialog box processing.

See Also

*Enterprise PeopleTools 8.50 PeopleBook: PeopleCode Developer's Guide*, "PeopleCode and the Component Processor"

Component Interface Attributes

This section discusses the name, keys, properties, collections, and methods of component interfaces.

Name

Each component interface requires a unique name that is specified when the component interface is created. The calling programs use the name of the component interface to access properties and methods.

Keys

Keys are special properties containing values that retrieve an instance (Get keys) or a list of instances (Find keys) of the component interface. When you create a new component interface, Get and Find keys are created based on the search record definition for the underlying component. However, you can add, remove, or change keys in PeopleSoft Application Designer. Create keys are created for components that have the Add action enabled.

Properties

Properties provide access to both component data and component interface settings. Component interfaces include two types of properties: standard and user-defined.

- Standard properties are assigned automatically when a component interface is created.
  
  Standard properties can be set to true or false. These properties are not displayed in the PeopleSoft Application Designer. Examples of standard properties include InteractiveMode, GetHistoryItems, and EditHistoryItems.

- User-defined properties map to record fields on the PeopleSoft component and are displayed in the PeopleSoft Application Designer.

  A property can correspond to a field or a scroll (collection). You can control which user-defined properties are included on the component interface.
Note. Every PeopleSoft Application Designer definition—including the component interface—has a
definition properties dialog box in which you make design-time settings for the definition. Those properties
should not be confused with the runtime properties that are discussed here.

Collections

A component interface collection is a special type of property that corresponds to a scroll. It contains fields
and subordinate scrolls as defined in its underlying component. By default, each collection uses the name of
the primary record for the underlying scroll.

Methods

A method is a function that performs a specific task on a component interface at runtime. As with component
interface properties, two main types of methods are available: standard and user-defined. For example, you
can use methods to save or create a new purchase order. Runtime access to each method is determined by the
security that you have for that specific method.

• Standard methods are those that are available for all component interfaces.

  The Find, Get, Save, and Cancel methods are automatically generated by PeopleSoft Application
  Designer when a new component interface is created. The Create method is created for components that
  have the Add action enabled. In the component interface designer, standard methods are highlighted in
  gray.

• User-defined methods are created in PeopleSoft Application Designer to provide added functionality to
  the component interface.

  These methods are functions that are made accessible through the component interface. Each function
  maps to a user-defined method. In the component interface designer, user-defined methods are
  highlighted in blue.

Component Interface Definitions and Views

You create, modify, and review your component interface definition by using PeopleSoft Application
Designer. You open the component interface definition just as you would any other definition, such as a page
or record.

When working with a component interface definition in PeopleSoft Application Designer, you see the
component view on the left and the component interface view on the right.

This example shows the component and component interface view in PeopleSoft Application Designer.
Component and component interface views in PeopleSoft Application Designer

The component view shows records and scrolls in the component, using a tree representation. The structure is the same as the one you see on the structure tab of a component in PeopleSoft Application Designer. Drag the fields and collections that you want exposed to the component interface view.

The component interface view shows the exposed keys, properties, and methods, using a tree representation. When you open a component interface, properties are displayed in the order in which they appear in the component view.

The tree nodes in both the component view and the component interface view have different icons. Some icons are used in both the component view and the component interface view with slightly different meanings. The following tables explain the meaning of each icon and column in the component interface view.

**Component Interface View Icons**

This table lists the component interface view icons:

- ![Component interface](image)
  - Component interface.

- ![Group of keys](image)
  - Group of keys.

- ![Property that is a key field from the underlying record](image)
  - Property that is a key field from the underlying record.
Alternate search key.

Group of properties or methods.

Collection.

Property or user-defined method.

Standard method.

Property that is a required field for the underlying record.

Item in a component interface that is no longer in sync with the underlying component. For example, if a field on which a property depends is deleted from the component, this icon appears.

**Component Interface View Columns**

The following terms describe the columns in the component interface view.

**Name**

Name of a specific element of a component interface (such as the name of a property or method). The default name for field properties is the field name. The default name for collections is the primary record name.

**Record**

Name of the underlying record on which a specific element is based. If the underlying record name changes, the component interface continues to point to the appropriate record.

**Field**

Name of the field to which a component interface property points. Like the record name, the underlying field name can change, and the component interface continues to point to the appropriate field.

**Read Only**

Y in this column indicates that a specific property has been marked read-only.

**Comment**

Identifies comments that exist in the Edit Property dialog box for the selected key, property, or collection.

---

**Note.** In the component interface view, properties appear in the same order as they appear in the component and are not sorted alphabetically.
Chapter 3

Developing Component Interfaces

This chapter discusses how to:

- Create component interface definitions.
- Use keys.
- Set properties.
- Work with collections.
- Work with methods.
- Validate component interfaces.
- Set security options.
- Test component interfaces.
- Understand synchronization.
- Write component interface programs.
- Understand runtime considerations.

Creating Component Interface Definitions

This section discusses how to:

- Create new component interface definitions.
- Name component interface definitions.
- Associate component interface definitions with menus.
- Determine the fields to expose in component interfaces.

Understanding Creating Component Interface Definitions

This section discusses key concepts for creating component interface definitions.
Component Structure

Because each component interface refers to a single component, you must know the structure of the component for which you are constructing a component interface. You can use an existing component within an application or create a new one for the sole purpose of constructing a component interface. Many parts of the component interface, such as the keys, are created based on settings in the referenced component.

Criteria for Setting Automatic Default Properties

To be able to set automatic defaults for fields in the new component interface, the system needs the properties to be of a specific field or page control type.

The fields should be of the following types:

- Character
- Long character
- Number
- Signed number
- Date
- Time
- Datetime

The field should be one of the following page control types:

- Edit box
- Drop-down list box
- Check box
- Radio button

The field cannot be invisible and should not be the same as the key field of the immediate parent.

Collections must have at least one child property that satisfies the field or page control criteria for providing the field by default. Collections with no properties are not added.

For a field on a secondary page to be selected for the default properties process, it must satisfy all the criteria for field type and page control and must be at the same level as the host page.

Additionally, the component tree that a component interface uses to order the properties lists the fields in the record based on their order in the record definition and not the order of the fields on the page. If the component tree lists the fields of a record based on the page, the properties of the component interface will reflect that order.

Creating New Component Interfaces

This section discusses how to create a new component interface.
New component interface with no properties yet defined.

To create a new component interface:

1. Select File, New from the PeopleSoft Application Designer menu.
2. Select Component Interface from the New dialog box.
3. Select the component on which to base this component interface.

After you select the appropriate component, you see a message asking whether you want the fields that are exposed in the selected component to become the default properties of the component interface.

Note. Not all fields on the component interface can have automatic defaults created for them.
4. Click Yes to confirm the default property definitions or No if you don't want any properties initially created.

If you elect to have the property definitions automatically provided by the system by default, then all properties that appear on the pages of the underlying component are added to the component interface. Even though the system adds the default properties, you may need to move other properties into the component view for the component to work.

An untitled component interface appears, showing the Get keys and Find keys. Create keys are produced only if the underlying component can run in Add mode (the example preceding this procedure does not have Create keys, because the search record of the underlying component cannot run in Add mode). PeopleSoft Application Designer generates the keys for you as you drag definitions.

The standard methods Cancel, Find, Get, and Save are automatically created. The Create method is not automatically created unless the component supports the Add mode.

**Note.** You can begin adding properties to a new component interface at any point. However, you cannot add any user-defined methods to the component interface until you have saved the component interface.

5. Save the component interface.

After you have saved the component interface, you can further define user-defined methods.

### Naming Component Interface Definitions

Like every other definition in PeopleTools, component interfaces must have unique names. The naming of component interfaces should be consistent and systematic. Also, the name should not be changed after the component interface is part of a production system—other applications depend on a consistent name with which to reference the component interface.

If you are changing the structure of a component interface such that an existing program can no longer access it correctly, create a new component interface rather than updating the existing one. No version property is on a component interface, so if you must create a new version of a delivered component interface, adhere to a standard naming guideline to avoid confusion. A suggested naming guideline is:

- LOCATION (original component interface).
- LOCATION_V2 (version two of the component interface).

### Associating Component Interfaces with Menus

This applies to component interfaces built from components that are already attached to one or more menus.
To associate a component interface with a menu:

1. Select File, Open from the PeopleSoft Application Designer menu to open an existing component interface.
2. Select File, Definition Properties from the PeopleSoft Application Designer menu.
   The Definition Properties dialog box appears.
3. Select the appropriate menu name on the General tab for this component interface.

**Note.** Associate a menu with a component interface only when PeopleCode is in the component that uses the %Menu system variable.
Determining the Fields to Expose in Component Interfaces

You expose fields from a component in the component interface by dragging a record field or a scroll from the component view into the component interface view. However, some forethought is required before exposing a component as a component interface. You need to have a thorough understanding of the underlying component so that you expose fields that are required in the external system. For example, if the component has a field called SSN, you need to be sure that the SSN field is required before exposing it to the external system. Expose only those properties that are necessary.

The component view displays fields that are available in the component buffer at runtime. For example, if a record containing 10 fields has only 3 fields included on a page, then the component view will list only those 3 fields.

The first time that you drag a scroll from the component view to the component interface view, the system uses the following rules to determine what properties to expose:

• Keys are exposed only at the highest-level collection in which they first appear.

  In some cases, this is not appropriate. When an effective-dated component that has the same level-zero and level-one record is exposed through a component interface, it should be exposed the same way in which it appears on a page in the component. In this case, only one key field typically appears at level zero and the effective-date keys appear at level one. The component interface wrapper should expose the page in the same fashion—removing keys that do not appear in the level-zero scroll from the component interface top-level collection and manually adding keys that appear in level-one scroll to the second-level collection.

  Typically, you do not want to expose Get keys or Create keys as properties, because these are set before a Get or Create operation and might be inadvertently changed.

• Make sure that you do not delete all the properties within the collection; that would result in an empty collection. If such empty collections exist, remove them; otherwise, they appear with X in the component interface view.

• If your page does not support Add mode, then you should not expose the level-zero record of the component, because it contains data that is not specific to the component interface that you are creating.

• Do not expose fields that are not visible in the component view.

  The component optimization code might eliminate unused fields from its buffers, which results in an error when that field is accessed by the component interface.

Using Keys

This section discusses how to add and delete keys.

Understanding Keys

The following table shows the three types of component interface keys:
### Key Type

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get keys</td>
<td>These keys automatically map to search key fields in the search record for the underlying component. You must change Get keys only if you modify the keys of the component after you create a component interface.</td>
</tr>
<tr>
<td>Find keys</td>
<td>These map to both search key fields and alternate search key fields in the search record for the underlying component. You can remove any Find keys based on alternate search key fields that you don't want to make available for searching.</td>
</tr>
<tr>
<td>Create keys</td>
<td>If the underlying component allows the Add action, then Create keys are generated for the component interface automatically. They map to fields marked as <em>Srch</em> (search) in the search record for the component (or the add search record, if one is specified).</td>
</tr>
</tbody>
</table>

Keys are created automatically when you create a component interface. Typically, you must manually add keys only if new search key fields are added to the underlying component after the creation of the component interface. However, you might want to modify the Find keys—either to restrict a user from searching on a particular key or to add an alternate search key that didn't exist when the component was created.

Component interface keys are based only on the search key fields and alternate search key fields that are designated as list box items in the search record of the underlying component. When you create the component interface, the keys are automatically generated from all key fields that qualify.

- Each search key field produces a Get key and a Find key.
- Each search key field also produces a Create key if the underlying component allows Add mode.
- Each alternate search key field produces a Find key.

### Valid Conditions for Modifying Keys

The following conditions are valid for modifying keys.

- You can add or delete a Find key if it is based on an alternate search key field.
- You can add any type of key based on a qualifying search key field in the component, if it isn't already the basis of an existing key of the same type.

  This is necessary only if a new search key field is added to the component after you create the component interface.

- You can delete any type of key if its underlying search key field meets one of these criteria:
  - It is no longer defined as a search key field.
  - It is no longer designated as a list box item.
  - It has been deleted from the component.
Note. An X icon precedes a name in the component interface view if the field underlying a component interface key no longer qualifies as a key. Remove keys (or any other properties) that are marked with this symbol to ensure proper operation of the component interface.

Adding and Deleting Keys

To add a key:

1. Expand the search key collection (the first collection) in the component view.
2. Drag the key to the component interface view.

To delete a key:

1. Select the key in the component interface view.
2. Press the Del key.

Setting Properties

This section provides an overview of standard properties and discusses how to:

• Create user-defined properties.
• Delete user-defined properties.
• Rename user-defined properties.
• Create reference properties.
• Make properties read-only.

Understanding Standard Properties

Standard properties do not appear in the component interface view in PeopleSoft Application Designer. The following tables name and define the standard properties, and list the interfaces for PeopleCode, Java, C++, and Visual Basic.

This table contains the component interface properties:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateKeyInfoCollection</td>
<td>Returns a collection of items that describes the Create keys. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: <code>IcompIntfcPropertyInfoCollection</code></td>
</tr>
<tr>
<td></td>
<td><code>getCreateKeyInfoCollection()</code></td>
</tr>
<tr>
<td></td>
<td>• C++: <code>HPSAPI_COMPINTFCPROPERTYINFOCOLLECTION &lt;CI_NAME&gt;_GetCreateKeyInfoCollection(HPSAPI_</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;CI_NAME&gt;</code>)</td>
</tr>
<tr>
<td></td>
<td>• COM: <code>CompIntfcPropertyInfoCollection</code></td>
</tr>
<tr>
<td></td>
<td><code>CreateKeyInfoCollection</code></td>
</tr>
<tr>
<td>GetKeyInfoCollection</td>
<td>Returns a collection of items that describes the Get keys. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: <code>IcompIntfcPropertyInfoCollection</code></td>
</tr>
<tr>
<td></td>
<td><code>getGetKeyInfoCollection()</code></td>
</tr>
<tr>
<td></td>
<td>• C++: <code>HPSAPI_COMPINTFCPROPERTYINFOCOLLECTION &lt;CI_NAME&gt;_GetGetKeyInfoCollection(HPSAPI_</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;CI_NAME&gt;</code>)</td>
</tr>
<tr>
<td></td>
<td>• COM: <code>CompIntfcPropertyInfoCollection</code></td>
</tr>
<tr>
<td></td>
<td><code>GetKeyInfoCollection</code></td>
</tr>
<tr>
<td>FindKeyInfoCollection</td>
<td>Returns a collection of items that describes the Find keys. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: <code>IcompIntfcPropertyInfoCollection</code></td>
</tr>
<tr>
<td></td>
<td><code>getFindKeyInfoCollection()</code></td>
</tr>
<tr>
<td></td>
<td>• C++: <code>HPSAPI_COMPINTFCPROPERTYINFOCOLLECTION &lt;CI_NAME&gt;_GetFindKeyInfoCollection(HPSAPI_</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;CI_NAME&gt;</code>)</td>
</tr>
<tr>
<td></td>
<td>• COM: <code>CompIntfcPropertyInfoCollection</code></td>
</tr>
<tr>
<td></td>
<td><code>FindKeyInfoCollection</code></td>
</tr>
<tr>
<td>Name</td>
<td>Description, Programming Syntax</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetHistoryItems</td>
<td>Controls whether the component interface runs in Update/Display mode or Correction mode when the underlying component is effective-dated. If GetHistory is set to true, then historical data can be retrieved but not modified. GetHistory items work in accordance with EditHistory items. The default value is False. This property is read-only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: boolean getGetHistoryItems(), void setGetHistoryItems(boolean)</td>
</tr>
<tr>
<td></td>
<td>• C++: BOOL \textlt{CI_NAME}_GetGetHistoryItems(HPSAPI_\textlt{CI_NAME}()), void \textlt{CI_NAME}_SetGetHistoryItems(HPSAPI_\textlt{CI_NAME}&gt;, BOOL)</td>
</tr>
<tr>
<td></td>
<td>• COM: Boolean GetHistoryItems</td>
</tr>
<tr>
<td>EditHistoryItems</td>
<td>Controls whether the component interface runs in Update/Display All mode. Update/Display mode, or Correction mode when the underlying component is effective-dated. If EditHistory items are set to true, then historical data can be modified. EditHistory items work in accordance with GetHistory items. The default value is False. This property is read-only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: boolean getEditHistoryItems(), void setEditHistoryItems(boolean)</td>
</tr>
<tr>
<td></td>
<td>• C++: BOOL \textlt{CI_NAME}_GetEditHistoryItems(HPSAPI_\textlt{CI_NAME}()), void \textlt{CI_NAME}_SetEditHistoryItems(HPSAPI_\textlt{CI_NAME}&gt;, BOOL)</td>
</tr>
<tr>
<td></td>
<td>• COM: Boolean EditHistoryItems</td>
</tr>
</tbody>
</table>
### Chapter 3 Developing Component Interfaces

#### InteractiveMode

Controls whether to apply values and run business rules immediately, or whether items are queued and business rules are run later, in a single step.

**Note.** You should use interactive mode when testing and debugging a component interface. Interactive mode in a production environment slows performance because of the number of server trips required.

If you are using a component interface as part of a batch process in which thousands of rows are to be inserted, running in interactive mode may reduce performance so much on some UNIX servers that the application times out with a connection failure.

The default value is False. This property is read-only.

Use these interfaces to call with other programming languages.

- **Java:** boolean getInteractiveMode(), void setInteractiveMode(boolean)
- **C++:** BOOL <CI_NAME>_GetInteractiveMode(HPSAPI_<CI_NAME>), void <CI_NAME>_SetInteractiveMode(HPSAPI_<CI_NAME>, BOOL)
- **COM:** Boolean InteractiveMode

#### StopOnFirstError

When this property is set to True, the first error generated by the component interface halts the program.

The default value is False. This property is read-only.

- **Java:** boolean getStopOnFirstError(), setStopOnFirstError(boolean)
- **C++:** BOOL <CI_NAME>_GetStopOnFirstError(HPSAPI_<CI_NAME>), void <CI_NAME>_SetStopOnFirstError(HPSAPI_<CI_NAME>, BOOL)
- **COM:** Boolean StopOnFirstError

<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>InteractiveMode</td>
<td>Controls whether to apply values and run business rules immediately, or whether items are queued and business rules are run later, in a single step. <strong>Note.</strong> You should use interactive mode when testing and debugging a component interface. Interactive mode in a production environment slows performance because of the number of server trips required. If you are using a component interface as part of a batch process in which thousands of rows are to be inserted, running in interactive mode may reduce performance so much on some UNIX servers that the application times out with a connection failure. The default value is False. This property is read-only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td>StopOnFirstError</td>
<td>When this property is set to True, the first error generated by the component interface halts the program. The default value is False. This property is read-only.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>InteractiveMode</td>
<td></td>
</tr>
<tr>
<td>StopOnFirstError</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description, Programming Syntax</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CompIntfcName</td>
<td>Returns the name of the component interface class as named in PeopleSoft Application Designer. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: String getCompIntfcName()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR</td>
</tr>
<tr>
<td></td>
<td>&lt;CI_NAME&gt;<em>GetCompIntfcName((HPSAPI</em>&lt;CI_NAME&gt;))</td>
</tr>
<tr>
<td></td>
<td>• COM: String GetCompIntfcName</td>
</tr>
<tr>
<td>ComponentName</td>
<td>Returns the name of the component interface class as named in PeopleSoft Application Designer. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: boolean GetComponentName()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR</td>
</tr>
<tr>
<td></td>
<td>&lt;CI_NAME&gt;<em>GetComponentName(HPSAPI</em>&lt;CI_NAME&gt;)</td>
</tr>
<tr>
<td></td>
<td>• COM: Boolean GetComponentName</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the description of the component interface class as set in PeopleSoft Application Designer. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: boolean getDescription()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR</td>
</tr>
<tr>
<td></td>
<td>&lt;CI_NAME&gt;<em>GetDescription((HPSAPI</em>&lt;CI_NAME&gt;))</td>
</tr>
<tr>
<td></td>
<td>• COM: String Description</td>
</tr>
<tr>
<td>Market</td>
<td>Returns the Market setting of the component used to build this component interface. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: String getMarket()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR</td>
</tr>
<tr>
<td></td>
<td>&lt;CI_NAME&gt;<em>GetMarket((HPSAPI</em>&lt;CI_NAME&gt;))</td>
</tr>
<tr>
<td></td>
<td>• COM: String Market</td>
</tr>
<tr>
<td>Name</td>
<td>Description, Programming Syntax</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>GetDummyRows</td>
<td>When a new scroll is inserted on a page, that scroll is displayed even though it has no underlying data. Any scroll that is empty has one dummy row displayed with only the defaults set. This property is True if the dummy row is to be displayed, False if it is not. The default value for this property is True. This property is read-write.</td>
</tr>
<tr>
<td></td>
<td>- Java: boolean getGetDummyRows(), void setGetDummyRows(boolean)</td>
</tr>
<tr>
<td></td>
<td>- C++: BOOL &lt;CI_NAME&gt;<em>GetGetDummyRows(HPSAPI</em>&lt;CI_NAME&gt;), void &lt;CI_NAME&gt;<em>SetGetDummyRows(HPSAPI</em>&lt;CI_NAME&gt;, BOOL)</td>
</tr>
<tr>
<td></td>
<td>- COM: Boolean GetDummyRows</td>
</tr>
<tr>
<td>PropertyInfoCollection</td>
<td>Returns a collection of items that describes a specific property. The specific properties that are available in the propertyinfocollection are listed here. This property is read-only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>- Java: IcompIntfcPropertyInfoCollection getPropertyInfoCollection()</td>
</tr>
<tr>
<td></td>
<td>- C++: HPSAPI_COMPINTFCPROPERTYINFOCOLLECTION &lt;CI_NAME&gt;<em>GetPropertyInfoCollection(HPSAPI</em>&lt;CI_NAME&gt;)</td>
</tr>
<tr>
<td></td>
<td>- COM: CompIntfcPropertyInfoCollection PropertyInfoCollection</td>
</tr>
</tbody>
</table>

The CompIntfPropInfoCollection object supports the following properties:

<table>
<thead>
<tr>
<th>PropertyName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>This property returns the name of the object executing the property as a string. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>- Java: String getName()</td>
</tr>
<tr>
<td></td>
<td>- C++: LPTSTR CompIntfcPropertyInfo.GetName(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>- COM: String name</td>
</tr>
<tr>
<td><strong>PropertyName</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>RecordName</td>
<td>This property returns the record name associated with the object executing the property. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: String getRecordName()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR CompIntfcPropertyInfo_GetRecordName(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: String RecordName</td>
</tr>
<tr>
<td>FieldName</td>
<td>This property returns the field name associated with the object executing the property. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: String getFieldName()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR CompIntfcPropertyInfo_GetFieldName(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: String FieldName</td>
</tr>
<tr>
<td>LabelLong</td>
<td>This property returns the record field Long Name value as a string. If a component override exists for this value, it is not included. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: String getLabelLong()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR CompIntfcPropertyInfo_GetLabelLong(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: String LabelLong</td>
</tr>
<tr>
<td>LabelShort</td>
<td>This property returns the record field ShortName value as a string. If a component override exists for this value, it is not included. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: String getLabelShort()</td>
</tr>
<tr>
<td></td>
<td>• C++: LPTSTR CompIntfcPropertyInfo_GetLabelShort(HPSAPI_CCOMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: String LabelShort</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IsCollection</td>
<td>This property returns True if the object executing the property is a data collection, False otherwise. If IsCollection is True, other field-oriented properties like Required, Type, Xlat, YesNo, Prompt, and Format are undefined. If IsCollection is False, the object represents a field and all the previous properties are defined as described. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>- Java: boolean getIsCollection()</td>
</tr>
<tr>
<td></td>
<td>- C++: BOOL CompIntfcPropertyInfo_GetIsCollection(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>- COM: Boolean IsCollection</td>
</tr>
<tr>
<td>Type</td>
<td>This property returns the field type, as a number, of the object.</td>
</tr>
<tr>
<td></td>
<td>This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>- Java: long getType()</td>
</tr>
<tr>
<td></td>
<td>- C++: PSI32 CompIntfcPropertyInfo_GetType(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>- COM: Long Type</td>
</tr>
<tr>
<td>OAType</td>
<td>This property returns the field type, as a number, of the object. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>- Java: long getOAType()</td>
</tr>
<tr>
<td></td>
<td>- C++: PSI32 CompIntfcPropertyInfo_GetOAType(HPSAPI_COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>- COM: Long OAType</td>
</tr>
<tr>
<td><strong>PropertyName</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| Format           | This property returns the field format for the object executing the property (that is, name, phone, zip, SSN, and so on) as a number. This property is read-only. See *Enterprise PeopleTools 8.50 PeopleBook: PeopleCode API Reference*, "Component Interface Classes," CompIntfPropInfoCollection Object Properties.  
  |  
  | Java: String getFormat()  
  | C++: PSI32  
  | CompIntfcPropertyInfo_GetFormat(HPSAPI.COMPINTFCPROPERTYINFO)  
  | COM: Long Format  
| Key              | This property returns True if the object executing the property is a key, False otherwise. This property is read-only.  
  |  
  | Java: boolean getKey()  
  | C++: BOOL  
  | CompIntfcPropertyInfo_GetKey(HPSAPI.COMPINTFCPROPERTYINFO hCompIntfcPropertyInfo)  
  | COM: Boolean Key  
| Required         | This property returns True if the object executing the property is a required property, False otherwise. This property is read-only.  
  |  
  | Java: boolean getRequired()  
  | C++: BOOL  
  | CompIntfcPropertyInfo_GetRequired(HPSAPI.COMPINTFCPROPERTYINFO)  
  | COM: Boolean Required  
| Xlat             | This property returns True if the object executing the property is associated with an XLAT table, False otherwise. This property is read-only.  
  |  
  | Java: String getXlat()  
  | C++: BOOL  
  | CompIntfcPropertyInfoGetXlat(HPSAPI.COMPINTFCPROPERTYINFO)  
<p>| COM: String Xlat  |</p>
<table>
<thead>
<tr>
<th>PropertyName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yesno</td>
<td>This property returns True if the object executing the property is associated with the Yes/No table, False otherwise. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: boolean getYesno()</td>
</tr>
<tr>
<td></td>
<td>• C++: BOOL CompIntfcPropertyInfo_GetYesno(HPSAPI_COMPI NTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: Boolean Yesno</td>
</tr>
<tr>
<td>Prompt</td>
<td>This property returns True if the object executing the property is associated with a prompt table, False otherwise. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: boolean getPrompt()</td>
</tr>
<tr>
<td></td>
<td>• C++: BOOL CompIntfcPropertyInfo_GetPrompt(HPSAPI_COMPI NTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: Boolean Prompt</td>
</tr>
<tr>
<td>Length</td>
<td>This property returns the length of the object executing the property. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: long getLength()</td>
</tr>
<tr>
<td></td>
<td>• C++: PSI32 CompIntfcPropertyInfo_GetLength(HPSAPI_COMPI NTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: Long Length</td>
</tr>
<tr>
<td>DecimalPosition</td>
<td>This property returns the decimal position for the object executing the property. This property is read-only.</td>
</tr>
<tr>
<td></td>
<td>• Java: long getDecimalPosition()</td>
</tr>
<tr>
<td></td>
<td>• C++: PSI32 CompIntfcPropertyInfo_GetDecimalPosition(HPSAPI _COMPINTFCPROPERTYINFO)</td>
</tr>
<tr>
<td></td>
<td>• COM: Long DecimalPosition</td>
</tr>
<tr>
<td>PropertyName</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| IsReadOnly   | This property returns True if the property marked read-only in the component interface definition; False otherwise. This property is read-only.  
  • Java: boolean getIsReadOnly()  
  • C++: BOOL CompIntfcPropertyInfo_GetIsReadOnly(HPSAPI_COMPINTFCPROPERTYINFO)  
  • COM: Boolean IsReadOnly |
| Altkey       | This property returns True if the object executing the property is an alternate key, False otherwise. This property is read-only.  
  • Java: boolean getAltkey()  
  • C++: BOOL CompIntfcPropertyInfo_GetAltkey(HPSAPI_COMPINTFCPROPERTYINFO)  
  • COM: Boolean Altkey |
| ListBoxitem  | This property returns True if the object executing the property is associated with a list box, False otherwise. This property is read-only.  
  • Java: boolean getListboxitem()  
  • C++: BOOL CompIntfcPropertyInfo_GetListboxitem(HPSAPI_COMPINTFCPROPERTYINFO)  
  • COM: Boolean ListBoxitem |

**Example of PropertyInfoCollection**

Here is a Java example that calls PropertyInfoCollection:
IcompIntfcPropertyInfoCollection oLO_PropInfoColl
IcompIntfcPropertyInfo oLO_PropInfoItem

oLO_PropInfoColl = oCI.getPropertyInfoCollection();
for (int I=0; I < oLO_PropInfoColl.getCount(); I++) {
  oLO_PropInfoItem = oLO_PropInfoColl.item(i);

  System.out.println("\t Name = " + oLO_PropInfoColl.getName());
  System.out.println("\t Record Name = " + oLO_PropInfoColl.getRecordName());
  System.out.println("\t Field Name = " + oLO_PropInfoColl.getFieldName());
  System.out.println("\t Label Long = " + oLO_PropInfoColl.getLabelLong());
  System.out.println("\t Label Short = " + oLO_PropInfoColl.getLabelShort());
  System.out.println("\t IsCollection = " + oLO_PropInfoColl.getIsCollection());
  System.out.println("\t Type = " + oLO_PropInfoColl.getType());
  System.out.println("\t OAType = " + oLO_PropInfoColl.getOAType());
  System.out.println("\t Format = " + oLO_PropInfoColl.getFormat());
  System.out.println("\t Is Get Key? = " + oLO_PropInfoColl.getKey());
  System.out.println("\t Is Required = " + oLO_PropInfoColl.getRequired());
  System.out.println("\t Is Xlat? = " + oLO_PropInfoColl.getXlat());
  System.out.println("\t Is Yesno? = " + oLO_PropInfoColl.getYesno());
  System.out.println("\t Prompt = " + oLO_PropInfoColl.getPrompt());
  System.out.println("\t Length = " + oLO_PropInfoColl.getLength());
  System.out.println("\t DecimalPosition = " + oLO_PropInfoColl.getDecimalPosition());
  System.out.println("\t Is Read Only? = " + oLO_PropInfoColl.getIsReadOnly());
  System.out.println("\t Is Alt Key? = " + oLO_PropInfoColl.getAltkey());
  System.out.println("\t Is ListBox item? = " + oLO_PropInfoColl.getListboxitem());
}

Object Adapter

The name of the property is OAType, and it holds the value of the object adapter type. Exposing this property and supplying the associated methods enables you to detect possible data type mismatches between the database and the component interface object.

The Java methods are:

getOAType() Returns the object adapter type.

getType() Returns the type of the property of a particular database field.

For example:
public static void printPropertyType(String propName, ICompIntfcPropertyInfo i⇒ PropertyInfo) {

    String strOAType = null;
    String strDBType = null;

    try {
        switch ((int)iPropertyInfo.getOAType()) {
            /* Object Adapter Type == 0 */
            case CIPropertyTypes.PSPROPERTY_OA_TYPE_BOOL:
                strOAType = "BOOL";
                break;
            /* Object Adapter Type == 1 */
            case CIPropertyTypes.PSPROPERTY_OA_TYPE_NUMBER:
                strOAType = "INTEGER";
                break;
            /* Object Adapter Type == 2 */
            case CIPropertyTypes.PSPROPERTY_OA_TYPE_FLOAT:
                strOAType = "FLOAT";
                break;
            /* Object Adapter Type == 3 */
            case CIPropertyTypes.PSPROPERTY_OA_TYPE_STRING:
                strOAType = "STRING";
                break;
        }

        switch ((int)iPropertyInfo.getType()) {
            /* Database Type == 0 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_CHARACTER:
                strDBType = "CHARACTER";
                break;
            /* Database Type == 1 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_LONG_CHARACTER:
                strDBType = "LONG CHARACTER";
                break;
            /* Database Type == 2 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_NUMBER:
                strDBType = "NUMBER";
                break;
            /* Database Type == 3 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_SIGNED_NUMBER:
                strDBType = "SIGNED NUMBER";
                break;
            /* Database Type == 4 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_DATE:
                strDBType = "DATE";
                break;
            /* Database Type == 5 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_TIME:
                strDBType = "TIME";
                break;
            /* Database Type == 6 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_DATETIME:
                strDBType = "DATETIME";
                break;
        }
    }

    catch (Exception e) {
        e.printStackTrace();
    }

    System.out.println("\n" + propName +
        " Object Adapter Type is: " + strOAType +
        " Database Type is: " + strDBType +
        " ");
}
Component Interface Collection Property

This table contains the component interface collection property Count.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Returns the number of items in a collection.</td>
</tr>
<tr>
<td></td>
<td>• Java: long getCount()</td>
</tr>
<tr>
<td></td>
<td>• C++: PSI32 CompIntfcCollection_GetCount(HPSAPI_&lt;CI_NAME&gt;)</td>
</tr>
<tr>
<td></td>
<td>• COM: Integer Count</td>
</tr>
</tbody>
</table>

Data Item Property

This table contains the data item property ItemNum:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemNum</td>
<td>Returns the position of the row within the given collection of a DataRow.</td>
</tr>
<tr>
<td></td>
<td>• Java: long getItemNum()</td>
</tr>
<tr>
<td></td>
<td>• C++: PSI32 &lt;CI_NAME&gt;<em>GetItemNum(HPSAPI</em>&lt;CI_NAME&gt;)</td>
</tr>
<tr>
<td></td>
<td>• COM: Integer ItemNum</td>
</tr>
</tbody>
</table>

Note. The component interface classes contain information about PropertyInfo properties and related PeopleCode.

Creating User-Defined Properties

User-defined properties are those properties on the underlying component that are exposed through the component interface. User-defined properties are derived from the component to which the component interface is associated and must be added manually. They are the specific record fields that you expose to an external system with the component interface. You create user-defined properties in addition to the standard properties to enable data manipulation of the component. When you create a new component interface, if you accept the default properties, user-defined properties are created automatically for each field displayed to the user on the underlying component.
User-defined properties are the points where the component and the underlying database are exposed to the external system. This is the means that component interfaces use to add or change fields and data in the database.

To create a user-defined property, drag a record, field, or scroll from the component view to the component interface view.

Where you insert the definition in the component interface view does not matter. When the component interface is opened, the system automatically converts the field or record into a component interface property and places it in the appropriate place in the list of properties. Also, when you drag a definition from the component view into the component interface view, all child definitions are brought into the component interface automatically. After these child properties are added to the component interface, you can remove each property individually, if desired.

Dragging a key from the search records, which precede the level-zero record in the page view, adds a key to all appropriate key collections (Get, Create, and Find) in the component interface. Because appropriate keys are added automatically when a component interface is first created, you typically must add keys only if the new keys are added to the underlying component after the creation of the component interface.

### Deleting User-Defined Properties

To delete a property:

1. Select the property to be deleted.
2. Either press the Del key on the keyboard, or right-click the key and select Delete.

   Standard Windows behavior is employed for selecting multiple properties using the Shift and Ctrl keys.

### Renaming User-Defined Properties

Property names are automatically generated according to the corresponding fields from the component. If these names are cryptic, you might want to rename these properties to explain them better. Renaming a property does not change the field that the property references.

**Important!** PeopleSoft Mobile Agent is a deprecated product. The options listed in the Mobile Property Persistence group box of the Edit Property dialog box exist for backward compatibility only.
To rename a property:

1. Double-click the property name or right-click the property name and select Edit Name from the menu.
2. Enter the new property name.

   Programs accessing this component interface must reference the new property name. For example, if SDK_NAME was changed to NAME, programs must use NAME instead of SDK_NAME.

3. Add any comments that might be helpful.

4. Select the Read-Only check box to make this property read-only.

5. If this property is for a mobile application, select a radio button that sets the persistence of the property.

   - Send Updates is the default behavior for a mobile property.
     
     Any changes or additions to this property on a mobile instance are synchronized to the server.

   - If a mobile property is set to Do not Send Updates, this property is not synchronized up to the server, but the value is maintained on the device.

   - A Derived property is used only at mobile runtime. Any values that are set or added to this property exist only for the runtime life of the object. No persistence of this data on the device exists, so it is subsequently never uploaded to the server.

---

**Note.** PeopleSoft Application Designer generates an error message if it detects that a component interface has properties that resolve to the same name when creating, saving, or opening a given component interface.

For example, NAME1 and NAME_1 both resolve to the same name when PeopleSoft APIs are built. The set and get functions that are generated for the properties RTE_CNTL_TYPE1 and RTE_CNTL_TYPE_1 are:

```java
public String getRteCntlType1()

public void setRteCntlType1(String inRteCntlType1)
```

This results in a compile error. To fix this condition, name the properties so that they do not resolve to the same name.

---

**Creating Reference Properties**

Each component interface is isolated and unaware of the other component interfaces in the system. To access and update information from other component interfaces, references establish relationships between component interfaces.

Create a reference property in one component interface to access data exposed in another component interface. For example, the Customer object and the component interface exposing its properties include properties such as the customer's name, address, and telephone numbers. Another object, Contact, includes data associated with all contacts in the system. The link between a specific customer and its associated contacts is owned by the Contact record, not the Customer record.

Therefore, to access contact data, the Customer component interface needs a reference property referring to the Contact component interface. For you to update contact data from the Customer component interface, the reference must include a valid reference path and reference backpointer to the customer ID.

Access the Create Reference dialog box by right-clicking the property and selecting Create Reference.
Create Reference dialog box

The Create Reference dialog box has the following fields:

**Name**
Describes the name of the reference you are creating.

**Related Component Interface**
Designates the component interface referenced from the current component interface.

**Comments**
Enter any comments to track the reference.

**Related Key Mapping**
Maps the property from the related component interface to the selected component interface property.
Valid Reference Path  Supports dynamic enumeration of the objects that can be selected as the value of the reference property being defined. This effectively filters these values so that you can select only objects that support the defined reference.

Because references use the concept of a walkpath to go from level zero of one component interface to level zero of another component interface, and then "walk" down to the lower levels of the component interface, only the level zero references are displayed in the Valid Reference Path drop-down list of a reference definition.

Reference Backpointer  Refers to the path back to the original component interface.

Making Properties Read-Only

You can make any property read-only. At runtime, the value of a read-only property can be read but not updated.

To make a property read-only:

1. Select the property.
2. Select Edit, Toggle Read Only Access from the PeopleSoft Application Designer menu.

A Y appears in the Read Only column of the component interface view corresponding to each property that you selected to be read-only.

Note. You can double-click the icon of any existing user-defined property to edit its name or comment or to toggle read-only access.

Working with Collections

A collection is a property that points to a scroll, rather than a field, in the underlying component for a component interface. A collection groups multiple fields in a scroll. All the fields in the scroll are mapped to a property. These properties are part of the collection.

You create collections the same way you create properties—drag the scroll from the component view into the component interface view. Consider these points when creating collections:

• When you drag a scroll into the component interface view, all child scrolls come with it.

This is the same behavior that you would expect when creating a property. Child properties are always added automatically when you drag a field from the component view to the component interface view. After the property or collection has been created, you can delete individual child properties or collections manually, if necessary.

• When you drag a scroll into the component interface view, all record fields contained in that scroll come with it—not just those from the record that defines the scroll.

The fields from all records at that scroll level are exposed as part of the same collection.
• Keys that appear in parent and child scrolls are not added to child collections.

For the component interface to function as expected, the keys must remain synchronized at all levels of the component. Having keys at lower levels makes compromising this synchronization possible. Therefore, lower-level keys are not introduced into the component interface and are not exposed to the user because those keys have already been set at the parent level.

• When you drag a child scroll into the component interface view, parent collections are created automatically.

For example, if you drag just the level-two scroll from the component view into the component interface view, a level-zero collection and a level-one collection are created automatically in the component interface. This hierarchy of collections is necessary so that you can navigate to the child collection at runtime.

---

### Working with Methods

This section provides and overview of session functions and methods, standard methods and collection methods. This section also discusses how to:

- Enable and disable standard methods.
- Create user-defined methods.
- Export user-defined methods.

### Understanding Session Functions and Methods

The session functions and methods connect to a session on an Application server. This connection must be made before you can use the component interface methods.

**Component Interface Session Functions**

This table contains the component interface session function `createSession`:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createSession</code></td>
<td>Returns a session object.</td>
</tr>
<tr>
<td></td>
<td>(In PeopleCode, &amp;session = %session)</td>
</tr>
<tr>
<td></td>
<td>• Java: ISession API.createSession()</td>
</tr>
<tr>
<td></td>
<td>• C++: HPSAPI_SESSION PSApiCreateSession()</td>
</tr>
<tr>
<td></td>
<td>• COM: PeopleSoft_PeopleSoft.Session CreateObject(“PeopleSoft.Session”)</td>
</tr>
</tbody>
</table>

**Component Interface Session Methods**

This table contains the component interface session methods:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connects to the application server. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td>(not used in PeopleCode)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Java: boolean connect(long apiVersion, string server, string username, string password, byte[] ExternalAuth)</td>
</tr>
<tr>
<td></td>
<td>• C++: Bool session_Connect(HPSAPI hSession, PSI32 ApiVersion, LPTSTR server, LPSTR username, LPTSTR password, PSAPIVAPBLOB ExternalAuth)</td>
</tr>
<tr>
<td></td>
<td>• COM: connect(apiVersion As Long, server As string, username As string, password As String, externalAuth As Integer) As Boolean</td>
</tr>
</tbody>
</table>

| getCompIntfc         | Returns a reference to a component interface. getCompIntfc also checks to determine whether the given user that is connecting has the appropriate security to access the component interface. Use these interfaces to call with other programming languages. |
|                      | • Java: I<CI_Name> getCompIntfc(string ciName)                                                                 |
|                      | • C++: HPSAPI_<CI_Name> Session_GetCompIntfc(HPSAPI_SESSION hsession, LPTSTR ciName)                           |
|                      | • COM: <CI_Name> GetCompIntfc(ciName As String)                                                             |

### Understanding Standard Methods

A method is a definition that performs a specific function on a component interface at runtime. Each standard method is added by default when the component interface is created and is available in PeopleCode and other programming languages. Like properties, methods are saved as part of a component interface definition. Two main types of methods are available: standard methods and user-defined methods.
<table>
<thead>
<tr>
<th><strong>Standard Methods</strong></th>
<th><strong>Description, Programming Syntax</strong></th>
</tr>
</thead>
</table>
| **Cancel**           | Backs out of the current component interface, canceling any changes made since the last save. This is equivalent to clicking the Return to Search button online. Returns True on success, and False on failure.
Use these interfaces to call with other programming languages.
- Java: boolean cancel()
- C++: BOOL <CI_NAME>_Cancel(HPSAPI_<CI_NAME> hObj)
- COM: Function Cancel() As Boolean |
| **Create**           | Creates a new instance of a component interface. This is equivalent to creating a new record in Add mode online. Returns True on success, and False on failure.
Use these interfaces to call with other programming languages.
- Java: boolean create()
- C++: BOOL <CI_NAME>_Create(HPSAPI_<CI_NAME> hObj)
- COM: Function Create() As Boolean |
| **Find**             | Performs a partial key search for a particular instance of a component interface, using the search keys at level 0. Returns a collection of component interface instances which match the search criteria. If no component interface instances match the search criteria, the count on the collection is zero.
Use these interfaces to call with other programming languages.
- Java: <CI_NAME>Collection find()
- C++: HPSAPI_<CI_NAME>COLLECTION <CI_NAME>_Find(HPSAPI_<CI_NAME> hObj)
- COM: Function Find() As <CI_NAME>Collection |
### Standard Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
</table>
| Get                  | Retrieves a particular instance of a component interface. This is equivalent to opening a record in Update/Display or Correction mode when online with a PeopleSoft application. Returns True on success, and False on failure. Use these interfaces to call with other programming languages.  
  - Java: boolean get()  
  - C++: BOOL  
    `<CI_NAME>_Save(HPSAPI_<CI_NAME> hObj)`  
  - COM: Function Get() As Boolean |
| Save                 | Saves an instance of a component interface. This is equivalent to clicking the Save button in the online system. Returns True on success, and False on failure. You should cancel after a save. Use these interfaces to call with other programming languages.  
  - Java: boolean save()  
  - C++: BOOL  
    `<CI_NAME>_Save(HPSAPI_<CI_NAME> hObj)`  
  - COM: Function Save() As Boolean |
| GetPropertyByName    | Returns the value of a property that is specified by name. This function typically is used only in applications that cannot get the names of the component interface properties until runtime. Use these interfaces to call with other programming languages.  
  - Java: Object getPropertyName(String str)  
  - C++: HPSAPI_OBJECT  
    `<CiCollectionItem>_GetPropertyByName(HPSAPI_<CI_COLLECTION_ITEM>_ hColItem, LPTSTR Name)`  
  - COM: Function GetPropertyByName(name As String) |
<table>
<thead>
<tr>
<th>Standard Methods</th>
<th>Description, Programming Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPropertyByName</td>
<td>Sets the value of a property that is specified by name. This function typically is used only in applications that cannot set the names of the component interface properties until runtime. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: long setPropertyByName(String str, Object o)</td>
</tr>
<tr>
<td></td>
<td>• C++: PSI32 &lt;CiCollectionItem&gt;<em>SetPropertyByName(HPSAPI</em>&lt;CI_COLLECTION_ITEM&gt; hColItem, LPTSTR name, HPSAPI_OBJECT Value)</td>
</tr>
<tr>
<td></td>
<td>• COM: Function SetPropertyByName(name As String, value) As Long</td>
</tr>
<tr>
<td>GetPropertyInfoByName</td>
<td>Returns specific information, such as length, about the definition of a property that is specified by name. This function typically is used only in applications that cannot get the names of component interface properties until runtime or by applications that need to provide a dynamic list of values that would normally be found in prompt tables. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td>(In PeopleCode, CompIntfPropInfoCollection)</td>
<td>• Java: IcompIntfcPropertyInfo getPropertyInfoByName(String name)</td>
</tr>
<tr>
<td></td>
<td>• C++: HPSAPI_COMPINTFCPROPERTYINFO&lt;CiPropOrItem&gt;<em>GetPropertyInfoByName(HPSAPI</em>&lt;CIPROPORITEM&gt; hPropOrItem, LPTSTR name)</td>
</tr>
<tr>
<td></td>
<td>where CiPropOrItem is the name of either a property or an item in a collection.</td>
</tr>
<tr>
<td></td>
<td>• COM: Function GetPropertyInfoByName(name As String) As CompIntfPropInfo</td>
</tr>
</tbody>
</table>

By default, each component interface is created with four standard methods—Cancel, Find, Get, and Save. Additionally, the Create standard method is generated if Create keys have been added to the component interface.
Example for GetPropertyInfoByName

The GetPropertyInfoByName method returns an object containing the property information. Here is a Java example that calls GetPropertyInfoByName:

```java
IcompIntfcPropertyInfo oCompIntfcPropertyInfo
oCompIntfcPropertyInfo = oCI.getPropertyInfoByName(tempName);
System.out.println(oCompIntfcPropertyInfo.getName());
if (!oCompIntfcPropertyInfo.getIsCollection()) {
    System.out.println("\t Format = " + oCompIntfcPropertyInfo.getFormat());
    System.out.println("\t Type = " + oCompIntfcPropertyInfo.getType());
}
System.out.println("\t Is Required = " + oCompIntfcPropertyInfo.getRequired());
System.out.println("\t Is Collection? = " + oCompIntfcPropertyInfo.getIsCollection());
System.out.println("\t Is Read Only? = " + oCompIntfcPropertyInfo.getIsReadOnly());
System.out.println("\t Is Get Key? = " + oCompIntfcPropertyInfo.getKey());
System.out.println("\t Label Long = " + oCompIntfcPropertyInfo.getLabelLong());
System.out.println("\t Label Short = " + oCompIntfcPropertyInfo.getLabelShort());
System.out.println("\t Length = " + oCompIntfcPropertyInfo.getLength());
System.out.println("\t Name = " + oCompIntfcPropertyInfo.getName());
System.out.println("\t Is Xlat? = " + oCompIntfcPropertyInfo.getXlat());
System.out.println("\t Is Yesno? = " + oCompIntfcPropertyInfo.getYesno());
```

Note. When creating a new component interface, you must save the component interface before the standard methods are created. PeopleSoft Application Designer adds the standard methods upon the first save of a new component interface.

Understanding Collection Methods

The first item in a component interface collection is always indexed as item 1 from PeopleCode and COM programs, which is consistent with other PeopleCode processing. From Java and C++ programs, this item is indexed as item 0.

Component Interface Collection Properties

This table contains the component interface collection properties:
<table>
<thead>
<tr>
<th>Data Collection Method</th>
<th>Action, Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Returns the number of items in a collection. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: long getCount()</td>
</tr>
<tr>
<td></td>
<td>• C++: PSI32 &lt;CiCollectionName&gt;<em>GetCount (HPSAPI</em>&lt;CI_COLLECTION_NAME&gt; hCol)</td>
</tr>
<tr>
<td></td>
<td>• COM: Count As Long</td>
</tr>
<tr>
<td>ItemByName (not used in PeopleCode)</td>
<td>Returns the property in the collection. It takes Name as a parameter. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: ICompIntfcPropertyInfo itemByName(String Name)</td>
</tr>
<tr>
<td></td>
<td>• C++: ComIntfcPropertyInfoCollection_ItemByName (HPSAPI_COMPINTFCPROPERTYINFOCOLLECTION, LPTSTR Name)</td>
</tr>
<tr>
<td></td>
<td>• COM: Function ItemByName(name As String) As ComIntfcPropertyInfo</td>
</tr>
<tr>
<td>InsertItem(Index)</td>
<td>Inserts a new item. This is equivalent to clicking the Add button to insert a new row when online. It takes Index as a parameter and follows the same conventions for performing business rules (PeopleCode) as the online system. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td></td>
<td>• Java: &lt;CiCollectionName&gt; insertItem( long Index)</td>
</tr>
<tr>
<td></td>
<td>• C++: HPSAPI_&lt;CLLECTION_ITEM&gt; &lt;CiCollectionName&gt;.<em>InsertItem(HPSAPI</em>&lt;CLLECTION_NAME&gt; hCol, PSI32 Index)</td>
</tr>
<tr>
<td></td>
<td>• COM: Function InsertItem(index As long) As &lt;CLLECTION_ITEM&gt;</td>
</tr>
</tbody>
</table>
### Data Collection Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Action, Usage</th>
</tr>
</thead>
</table>
| DeleteItem(Index) | Deletes the item that is designated by *Index*. This is equivalent to clicking the Delete button to delete the selected row when online. Use these interfaces to call with other programming languages.                  | Java: boolean deleteItem(long Index)  
C++: BOOL <CiCollectionName>_DeleteItem(HPSAPI_<CI_COLLECTION_NAME> hCol, PSI32 Index)  
COM: Function DeleteItem(index As Long) As Boolean                                                                                                   |
| Item(Index)     | Takes an item number as a parameter and returns a definition of the type that is stored in the specified row in the collection. For example, if the collection is a data collection, the return value is a DataRow. If the return value is a PropertyInfoCollection, then the return value is a PropertyInfo definition, and so on. Use these interfaces to call with other programming languages. | Java: <CiCollectionName> item(long Index)  
C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_Item(HPSAPI_<CI_COLLECTION_NAME> hCol, PSI32 Index)  
COM: Function Item(item As Long) As <CI_COLLECTION_ITEM>                                                                                           |
| ItemByKeys(keys) | Identifies and finds a specific item, based on keys. The keys vary according to the design of the collection. Use these interfaces to call with other programming languages.                                               | Java: <CiCollectionName> itemByKeys(String Key1, String Key2, …)  
C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_ItemByKeys(HPSAPI_<CI_COLLECTION_NAME> hCol, LPTSTR Key1, LPTSTR Key2, …)  
COM: Function ItemByKeys(KEY_1 As String, KEY_2,…) As <CI_COLLECTION_ITEM>                                                                            |
### Data Collection Method

<table>
<thead>
<tr>
<th>Data Collection Method</th>
<th>Action, Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CurrentItem</strong></td>
<td>Returns the current effective DataRow in the collection. The behavior is consistent with effective date rules that are used online. This method works with effective-dated records only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td><strong>CurrentItemNum</strong></td>
<td>Returns the item number of the current effective DataRow in the collection. The behavior is consistent with effective date rules that are used online. This method works with effective-dated records only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td><strong>GetEffectiveItem</strong></td>
<td>Returns the DataRow that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItem function, which returns the definition that is effective at this moment. This method works with effective-dated records only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td>Data Collection Method</td>
<td>Action, Usage</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>GetEffectiveItemNum(DateString, SeqNum)</td>
<td>Returns the item number of the DataRow in the collection that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItemNum function, which returns the number of the definition that is effective at this moment. This method works with effective-dated records only. Use these interfaces to call with other programming languages.</td>
</tr>
<tr>
<td>• Java: long getEffectiveItemNum(string Date, long SeqNum)</td>
<td></td>
</tr>
<tr>
<td>• C++: &lt;CiCollectionName&gt;<em>GetEffectiveItemNum(HPSAPI</em>&lt;CI_COLLECTION_NAME&gt; hCol, LPTSTR Date, PSI32 SeqNum)</td>
<td></td>
</tr>
<tr>
<td>• COM: Function GetEffectiveItemNum(Date As String, SeqNum As Long)</td>
<td></td>
</tr>
</tbody>
</table>

**Enabling and Disabling Standard Methods**

You can control whether standard methods are accessible at runtime.
Enabling standard methods for runtime accessibility on the Properties-Standard Methods tab

To enable or disable standard methods:

1. Select File, Definition Properties from the PeopleSoft Application Designer menu.
   
   The Definition Properties dialog box appears.

2. Select the Standard Methods tab.

   You can enable or disable any of the standard methods selecting the corresponding check box. Doing so determines whether the method is available at runtime when the component interface is accessed. The Create option is available only if the component interface has Create keys.

Creating User-Defined Methods

This section discusses how to create user-define methods.
Creating user-defined methods in PeopleCode

To create a user-defined method:

1. Right-click anywhere in the component interface view.

2. Select View PeopleCode from the menu.

   The PeopleCode editor appears. If you are using a new component interface, no PeopleCode will appear in the editor because no user-defined methods have been created.

3. Write the required PeopleCode functions.

   PeopleCode functions that you write are stored in a single PeopleCode program that is attached to the component interface and associated with the Methods event.

   Note. New user-defined methods do not appear in the list of methods until you save the component interface. Double-click the icon of any existing user-defined method to return to this PeopleCode program.

4. Set permissions for the methods that you created.

   You must set permissions for every user-defined method. If you set permission to Full Access, at runtime that function is exposed to external systems as a method on the component interface object.
Exporting User-Defined Methods

If you want a user-defined component interface to be exportable, meaning used by code that instantiates the component interface, the method PeopleCode definition must include a Doc statement. It is in the form of:

```
Doc <documentation for method>
```

where `<documentation for method>` describes what the method does

For example, the following method returns true if `foo` is positive; otherwise, it returns false.

```
Function MyFooBar(int foo) returns boolean
Doc
if (foo >0) then
return True;
else
return False;
end-if;
end-function;
```

Validating Component Interfaces

Validation ensures that the structure of a component interface is still valid. Over time, the structure of a component interface can become invalid due to component structural changes and modifications. For example, this can happen whenever a component deletes or adds a record or field. It can also happen if the keys on the component are added or removed. Properties and keys that no longer synchronize with their associated components are marked with an X icon.

**Note.** The validation process determines only whether the underlying component of a component interface has changed. It does not validate the PeopleCode that is associated with a component interface. To validate the PeopleCode, open the component and select Tools, Validate from the PeopleSoft Application Designer menu.

To correct an invalid component interface, you might have to delete properties for which appropriate fields or records no longer exist. If the structure of the source component has changed, you might have to delete old properties and re-add the new properties in their appropriate locations. You may also need to rename a property or collection.

To validate a component interface:

1. Open the component interface in PeopleSoft Application Designer.
   Validation occurs automatically whenever you open a component interface in PeopleSoft Application Designer.
2. Select Tools, Validate for Consistency from the PeopleSoft Application Designer menu to validate an open component interface.

As you change components or other related definitions, you should validate a component interface that is already open in PeopleSoft Application Designer.
Note. PeopleSoft Application Designer also validates each component interface upon its creation.

If a component interface definition becomes invalid, you cannot save changes to it in PeopleSoft Application Designer.

If a component interface definition is associated with an active Integration Broker service, you cannot delete it.

Setting Security Options

After creating a component interface, you must set security for it. Each individual method also needs to be provided security. Security for the component interface is provided through the PeopleSoft Internet Architecture pages. Component interface permissions are set at the permission list level in PeopleSoft security.

Component Interface Permissions

<table>
<thead>
<tr>
<th>Component Interface Permission</th>
<th>*Method Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>Full Access</td>
</tr>
<tr>
<td>Create</td>
<td>Full Access</td>
</tr>
<tr>
<td>Find</td>
<td>Full Access</td>
</tr>
<tr>
<td>Get</td>
<td>Full Access</td>
</tr>
<tr>
<td>Save</td>
<td>Full Access</td>
</tr>
</tbody>
</table>

Setting access permissions for methods

To set up component interface security:

1. Sign in to the PeopleSoft Pure Internet Architecture through the browser, and select PeopleTools, Security, Permissions & Roles, Permission Lists.

2. Select the permission list for which you want to set security.

   The Permission List component appears.

3. Access the Component Interfaces page.

4. Select the component interface for which you want to set security.

   To add another component interface to the list, click the Add button.
5. Click Edit.

The Component Interface Permissions page appears, showing all of the methods (both standard and user-defined) in the component interface and their method of access.

6. Set the access permission for each method.

Select Full Access or No Access. You must grant full access to at least one method to make the component interface available for testing and other online use.

7. Click OK when you are done.

8. Save the page.

Testing Component Interfaces

After setting the security for a component interface, you can test the contents and behavior using the component interface tester. You should test the component interface before using it in your external system. This proactive tool helps you discover problems with the underlying component or the component interface itself, including user-defined methods. When you are testing a component interface, real data from the database is used. Therefore, if you save the information that you change by calling the Save method, the information is changed in the database.

With the component interface tester, you can:

- Test the component interface in interactive mode.
- Retrieve history items.
- Test the standard, custom, and collection methods.

Searching Component Interfaces to Test

To test the component interface, you search for the component interface to test, and then you test it.

Access the Component Interface Tester search dialog box:
Chapter 3 Developing Component Interfaces

To search for a component interface to test:

1. Open the component interface in PeopleSoft Application Designer.
2. Select Tools, Test Component Interface from the PeopleSoft Application Designer menu.

   The Component Interface Tester search dialog box appears. This dialog box displays the keys (in the left-hand columns) for getting, creating, or finding an instance of the component interface. The right-hand columns provide a place for you to enter sample key values for testing.

3. Enter key values.
   a. Double-click the column to the right of any displayed keys.
   b. Enter the value in the right-hand column.

   The data that is used for the test corresponds to the key values that you enter here. In the preceding example, we have entered an employee ID of 6602.
Interactive Mode

In interactive mode, any action request occurs immediately. Each property being set causes an immediate trip to the application server (or database server in two-tier mode). This differs from noninteractive mode, in which actions are often held and later sent in batches. For example, in noninteractive mode, if you set a property, the property is not validated until you perform the save. However, in interactive mode the property is validated immediately. This means that edit processing (and other processing, such as FieldChange PeopleCode) occurs for each set property.

Whether you select this option depends on how you expect a particular component interface to be used and what you are currently testing. In a real production system, this parameter can significantly affect performance, but it makes little difference in the test component. In noninteractive mode, errors and properties are not updated until a method is run. By default, Interactive Mode is selected in the component interface tester.

Get History Items

Select to retrieves history data. This option applies to effective-dated fields only and is equivalent to running in either Update/Display or Update/Display All mode.

Edit History Items

Select to enable editing and saving of history data. This option applies to effective-dated fields only and is equivalent to running in either Update/Display or Correction mode.

Get Dummy Rows

Specify whether to get dummy rows. This option is selected as a default.

The component processor provides dummy rows to enable quick data entry when the level you are accessing does not have any data. Because of this, an API that does not need this row finds it and exposes it to the user. The application that uses the API now has to determine whether the row is a dummy row and accordingly decides to execute Item or InsertItem.

Setting the GetDummyRow to false enables the component interface processor to handle the counts accordingly. With this property set to false, users do not have to use item and InsertItem when adding new data at levels 1 to 3. Instead, they can comfortably always use InsertItem.

Get Existing

Clicking Get Existing is equivalent to opening a record in Update/Display or Correction mode online. It retrieves one instance from the database. After you click the Get Existing button, the Component Interface Tester dialog box appears.

Create New

Clicking Create New is equivalent to creating a new row in Add mode online. If your component does not support the Create method, this button is disabled. After you click the Create New button, the Component Interface Tester dialog box appears.

Getting Existing Records by Using Partial Keys

If you want to retrieve a partial key, click the Find button on the Component Interface Tester page. The Find Results dialog box appears:
Component Interface Tester — Find Results dialog box

You can choose the specific instance by selecting and clicking the Get Selected button. If you do not enter a partial key before clicking Find, all key values in the database are returned (subject to the maximum count of 300, just as when online). This is the same as calling the Find method through the component interface API, followed by selecting a value from the Find results, setting the Get key, and calling the Get method. After you click the Get Selected button, the Component Interface Tester dialog box appears.

Testing Component Interfaces

After you have searched for and retrieved the component interface, the Component Interface Tester dialog box appears.
Testing Component Interface Properties

From the Component Interface Tester dialog box, change the value of a property, double-click a value, and enter a new value. Some basic validation is done when you leave the field, which is equivalent to leaving a field using the Tab key in the online case. This validation includes system edit, FieldChange PeopleCode events, and FieldEdit PeopleCode events. Further validation can be done when the Save method is called (SaveEdit, SavePreChange, Workflow, and SavePostChange). If errors occur or warnings are encountered, messages are displayed in the Error Message Log area at the bottom of the window. The error message log displays the same text that would appear in the PSMessages collection of the Session object if you accessed the component through the Component Interface API.

Testing Component Interface Methods

Test component interface methods by right-clicking the component interface name.

A menu appears that shows the Save and Cancel standard methods and any user-defined methods that exist for the component interface. The Find, Create, and Get standard methods are not valid for an instantiated component, and therefore are not shown.

If a component interface method requires one or more parameters, a dialog box in which you can enter the parameters appears. After the method is executed, the same dialog box appears again, displaying changes to the parameters that were caused by the method. The return value of the function appears in the title of the dialog box. If a component interface requires no parameters, you do not see the initial dialog box, but you do see the return value dialog box following the function call.
Note. Because running a component interface method can result in a change to the component interface structure, PeopleSoft Application Designer always redraws the component interface tree in its collapsed form following a method call.

Testing Collection Methods

Test collection methods by right-clicking the collection name.

A menu appears that shows the standard collection methods. Select the collection method that you want to test for this component interface. After you select a collection method to test, the Enter parameters dialog box prompts you to enter an item number for the collection method that you are testing. The value that you enter for index [Number] is used to retrieve, insert, or delete an item, according to the following rules.

After you enter an index number, the result appears in the dialog box. If a return value is sent, it is displayed in the title bar. Otherwise, the message No value is displayed. Click OK or Cancel to dismiss the dialog box.

Collection Method Rules

This table contains the collection method rules:

<table>
<thead>
<tr>
<th>Collection Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item(index)</td>
<td>Returns the row at the specified index. Only the success or failure of this routine is of interest from inside the test component.</td>
</tr>
<tr>
<td>InsertItem(index)</td>
<td>Inserts a new row either before the index that you specify if the collection is effective-dated or following the index if it isn't effective-dated.</td>
</tr>
<tr>
<td>Collection Method</td>
<td>Purpose</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>DeleteItem(index)</td>
<td>Deletes the row that is designated by the index number that you specified in the Enter parameters dialog box.</td>
</tr>
<tr>
<td>ItemByKeys(key1, key2, …)</td>
<td>Returns the row corresponding to the specified keys. Only the success or failure of this routine is of interest from inside the test component.</td>
</tr>
<tr>
<td>CurrentItem</td>
<td>This method returns the effective row in an effective-dated record. Only the success or failure of this routine is of interest from inside the test component.</td>
</tr>
<tr>
<td>GetEffectiveItem(DateString, SeqNum)</td>
<td>Returns the data row that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItem function, which returns the definition that is effective at this moment. This method works with effective-dated records only.</td>
</tr>
<tr>
<td>GetEffectiveItemNum(DateString, SeqNum)</td>
<td>Returns the item number inside the collection of the data row that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItemNum function, which returns the number of the definition that is effective at this moment. This method works with effective-dated records only.</td>
</tr>
</tbody>
</table>

**Note.** Component interface classes contain information about collection methods.

**Determining ItemByKeys Parameters**

You can get the signature for the ItemByKeys method (or any other method) when testing a component interface. This is helpful for the ItemByKeys method, because its signature is different for each component interface.
Viewing the signature of the ItemByKeys method

To determine ItemByKeys parameters:

1. Open the definition.
2. Select Tools, Test Component Interface.
3. Find or get an appropriate populated component interface.
4. Navigate to the appropriate collection.
5. Right-click, and select ItemByKeys from the menu.

A dialog box appears, showing the specific parameters and types and the order in which you should call ItemByKeys.

In the preceding example, the keys for the SDK_BUS_EXP_PER ItemByKeys method are SDK_EMPID (String) and SDK_EXP_PER_DT (String).

---

**Understanding Synchronization**

The Component Interface Properties Synchronization tab is used with PeopleSoft Mobile Agent.

**Important!** PeopleSoft Mobile Agent is a deprecated product. The Component Interface Properties Synchronization tab exists for backward compatibility only.

PeopleSoft Mobile Agent extends the functionality of PeopleSoft Pure Internet Architecture to disconnected mobile devices, enabling users to continue working with their PeopleSoft applications on a laptop computer or personal digital assistant (PDA) while disconnected from the internet or local network.
Writing Component Interface Programs

The following chapters in this PeopleBook describe how to write component interface programs in several programming languages.

Also, the PeopleTools PeopleCode Reference contains a chapter that describes the component interface classes, including detailed instructions on the life cycle of a component interface and how to implement a component interface program in PeopleCode. You can use this information to help design your component interface program in other programming languages.

See Enterprise PeopleTools 8.50 PeopleBook: PeopleCode API Reference, "Component Interface Classes."
Understanding Runtime Considerations

In many ways, accessing a component interface is functionally equivalent to working with an online component. However, some important differences exist between component interfaces and components. This section describes how those differences affect interactive operation, functionality designed for graphical interfaces, client versus server operation, and several miscellaneous situations. These considerations, unless otherwise noted, apply to all the programming languages listed in this manual.

General Considerations

This section discusses general considerations for component interface programs.

WinMessage Unavailable

You cannot use the WinMessage API in a component that will be used to build a component interface. Use the MsgGet() function instead.

Email from a Component Interface

To use a component interface to send email, use the TriggerBusinessEvent PeopleCode event, not SendMail.

Related Display

Related display fields are not available for use in a component interface because they are not held in the buffer context that the component interface uses.

Row Inserts

If row insert functionality has been disabled for a page, you must take care when calling inserts against the corresponding component interface. Any PeopleCode associated with buttons used on the page to add rows will not be invoked by the component interface when an insert is done.

Note. If a component has logic that inserts rows on using the RowInsert event, the component interface cannot identify the change and locate the rows that were inserted by the application code. Generic interfaces such as Excel to Component Interfaces utility and the WSDLToCI will not function correctly when using this type of dynamic insert.

Scope Conflicts

This section discusses scope conflicts for component interface programs.

Infinite Processing Loops

A component interface should not call itself in any of the PeopleCode included within its component definition, because this may result in an infinite loop of the component interface. A component interface also should not call itself from a user-defined method.
**Multiple Instances of a Component Interface**

Because of potential memory conflicts, COM or C++ applications should not create multiple, simultaneous instances of the same component interface, either within a single procedure, or in both a parent and a child procedure.

**Interactive Mode**

This section discusses interactive mode considerations for component interface programs.

**UNIX Server Performance**

If you are using a component interface as part of a batch process in which thousands of rows are being inserted, running in interactive mode may reduce performance enough on some UNIX servers to produce a connection failure. Prevent this by setting the InteractiveMode property to False.

**Hidden Edit Validation Errors**

If the InteractiveMode property is set to True, and if a transaction sets a property to a value that isn't allowed in a prompt edit field, the edit field value is reset back to its original value. The error is logged in the PSMessages collection; however, the Save method runs without errors. Check the value of both the Save method and the collection ErrorPending property to discover all of the errors.
Chapter 4

Programming Component Interfaces
Using PeopleCode

This chapter provides an overview of PeopleCode behavior and limitations and discusses how to:

• Generate a PeopleCode template.
• Understand the PeopleCode template.

Understanding PeopleCode Behavior and Limitations

Note the behavior and limitations discussed in this section when you write PeopleCode for a component interface.

PeopleCode Event and Function Behavior

PeopleCode events and functions that relate exclusively to the graphical user interface and online processing cannot be used by component interfaces. These include:

• Search dialog processing.

  When you run a component interface, the SearchInit, SearchSave, and RowSelect events do not fire. This means that any PeopleCode associated with these events will not run. The first event to run is RowInit.

• Menu PeopleCode and pop-up menus.

  The ItemSelected and PrePopup PeopleCode events are not supported. In addition, the CheckMenuItem, DisableMenuItem, EnableMenuItem, HideMenuItem, and UncheckMenuItem functions are not available.

• Transfers between components, including modal transfers.

  The TransferPage, DoModalPageGroup, and IsModalPageGroup functions cannot be used.

• Dynamic tree controls.

  Functions related to this control, such as GetSelectedTreeNode, GetTreeNodeParent, GetTreeRecordName, RefreshTree, and TreeDetailInNode cannot be used.

• ActiveX controls.

  The PSControlInit and PSLostFocus events are not supported, and the GetControl function cannot be used.
• DoSave() and DoSaveNow().

The DoSave() and DoSaveNow() pcode functions are not supported. You should use the component interface Save() method and wrap the DoSave() and DoSaveNow() functions so that they do not execute when called from a component interface.

• Functions that are ignored in a component interface call.

Some PeopleCode functions are ignored if they are called through a component interface. These functions are:

  • WinMessage
  • CheckMenuItem
  • DisableMenuItem
  • EnableMenuItem
  • HideMenuItem
  • UncheckMenuItem
  • SetCursorPos
  • TransferPanel
  • TransferPage
  • DoModalComponent
  • IsModalComponent
  • DoModalPanelGroup
  • IsModalPanelGroup
  • GetSelectedTreeNode
  • GetTreeNodeParent
  • RefreshTree
  • TreeDetailInNode
  • GetControl
  • DoSave
  • DoSaveNow
  • Gray
  • Ungray
CopyRowset Language Considerations

In previous PeopleSoft releases, CopyRowset* functions for component interfaces were not sensitive to the language code on PSCAMA. Because of this, related language processing did not take place when language code on PSCAMA was different from the base language code. PeopleSoft now detects the language code in PSCAMA.

Limitations of Client-Only PeopleCode

Component interfaces can run on either the client or the server. By default, a component interface runs on the server. It runs on the client only if the code calling the component interface is running on a client machine.

Component interfaces must run either entirely on the server or entirely on the client. To ensure this runtime restriction, component interface references declared in PeopleCode must be declared as local, not global, variables.

Some built-in functions are always client-only; others are client-only under specific conditions.

Some built-in functions behave differently when used in three-tier mode, as opposed to two-tier mode.

Generating PeopleCode Templates to Access Component Interfaces

To access a component interface using PeopleCode, PeopleSoft Application Designer generates a template in the form of boilerplate PeopleCode that you can adapt to your purposes. This section describes how to generate the template code.
PeopleCode generated by dragging and dropping a component interface

To generate a PeopleCode template for a component interface:

1. Open the desired component interface definition in PeopleSoft Application Designer.
2. Insert the component interface into a project.
   a. Select Insert, Current Object into Project.
   b. Save the project.
3. Open the PeopleCode editor.
   You can associate component interface PeopleCode with a record, a component, a service operation handler, or an Application Engine program.
4. Select the component interface from the project workspace.
   Drag and drop the object from the project into the PeopleCode Editor.
5. Make any necessary changes to the PeopleCode in the PeopleCode Editor window.
Understanding PeopleCode Templates

The code shown in this section is a dynamically generated PeopleCode template that you can use as a starting point. Replace all default values or <*> notations with specific values or references to valid PeopleCode variables (replace this entire three-character string: <*>).

**Note.** The requirement to populate a non-create key is no longer a requirement to do the initial save.

PeopleCode runs only if you are connected. This means that you do not have to explicitly connect. Instead, connect to the existing session, using the %Session system variable.


You cannot connect to a different database through PeopleCode.

Set the PeopleSoft session error message mode. This property is used to determine how messages are output. This property takes either a numeric value or a constant. The default value is 1 (%PSMessages_CollectionOnly).

This property sets the value for the session. You can change modes during a session, for example, if you're starting a component interface. However, after you run the component interface, you should set the value back. Here is the list of modes that you can use:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Return no messages.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Default. Log messages into the PSMessage collection.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Display a pop-up message or dialog box.</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Log messages into the PSMessage collection and pop up a message dialog box.</td>
</tr>
</tbody>
</table>


**PeopleCode Template Notes**

Get a reference to the component interface providing its name. (A runtime error occurs if the component interface does not exist.)

Set the keys for the component interface. In this example SDK_EMPLID is the Get key.

The get() method retrieves data from the database, associated with the key values.

Get and print properties at level 0.
Similar code is generated for the properties SDK_BIRTHDATE and SDK_DEPTID.

Get collection at level 1 (SDK_BUS_EXP_PER).

Get and print properties at level 1.

Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the SDK_BUS_EXP_PER collection.

Get collection at level 2 (SDK_BUS_EXP_DTL).

&oSdkBusExpDtlCollection = &oSdkBusExpPer.SDK_BUS_EXP_DTL;

Get and print properties at level 2.

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD, SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.
Chapter 5

Programming Component Interfaces in Java

This chapter discusses how to:

• Build APIs in Java.
• Set up the Java environment.
• Generate Java runtime code templates.
• Understand generated Java code.

Building APIs in Java

If you plan to access your component interface from a Java external application, you must create a component interface API. The APIs are in the form of *.java source code files, which should be compiled into Java classes.

To build the component interface bindings:

1. Open any component interface definition in PeopleSoft Application Designer.

   Use any component interface definition, because you can build APIs for all of them, regardless of which one is open.

2. Select Build, PeopleSoft APIs.

   The Build PeopleSoft API Bindings dialog box appears.

3. Select the Build check box in the Java Classes group box.

   For the target directory, enter the directory in which you want the Java class source files to be created.

4. Click OK to build the bindings that you selected.

   The files that constitute the bindings are built in the location that you specified. If the operation is successful, a Done message appears in the PeopleSoft Application Designer Build window.
5. Compile the APIs that you just generated.

You could use these commands:

```bash
cd %PS_HOME%\class\PeopleSoft\Generated\CompIntfc
javac −classpath %PS_HOME%\class\psjoa.jar *.java
cd c:\pt8\class\PeopleSoft\Generated\PeopleSoft
javac −classpath %PS_HOME%\class\psjoa.jar *.java
```

---

**Setting Up the Java Environment**

When deploying component interfaces on a local client machine or web server with Java bindings, you must have:

- The third-party Java application.
- The application server and database.
- The Java Virtual Machine (JVM) supplied with Sun Microsystems Java Development Kit (JDK). The JDK may already be installed on your system. To verify that the JVM is installed, check the `%PS_HOME%\JRE` directory. If it is not installed, you can obtain download information at the Sun Microsystems web site.


To set up your client machine to access the component interface API using Java:

1. If it is not already installed on your system, install the Sun Microsystems JDK to enable the JVM.

   You can download the JDK to any location, for example `c:\bea\jkd<version>`.

2. Set the environment variable PATH to include the directory containing `jvm.dll`.

   For example, you might set it at `c:\bea\jkd<version>\jre\bin\client`; or, if the PeopleTools install is done locally, the path is `<PS_HOME>\jre\bin\client`.

3. Set the environment variable CLASSPATH to include:

   - The file `psjoa.jar` (typically `<PS_HOME>\class\psjoa.jar`).
   - The target directory selected during the Build API process (`<PS_HOME>\class`).

**Note.** In previous releases, sites using UNIX servers received the following error message when invoking a component interface through the PeopleSoft Java Object Adapter (PSJOA): *PSProperties not loaded from file.* To resolve this issue, copy the `pstrools.properties` file to the component interface execution directory.

---

**Generating Java Runtime Code Templates**

To access a component interface through external APIs using Java, PeopleSoft Application Designer generates a template in the form of boilerplate Java code that you can adapt to your purposes.
External Java APIs are located in the `<PS_CFG_HOME>\ExtAPI_Java` directory.

This section describes how to generate the template code.

Generating Java template

To generate a Java template for a component interface:

1. Open a component interface definition in PeopleSoft Application Designer.
2. Right-click anywhere in the definition view to display the menu.
3. Select Generate Java Template.

When the template is successfully generated, a message appears stating the name and location of the template file.

**Note.** The template file is generated in the directory specified by the TEMP or TMP system environment variable on your client machine.

4. Edit the generated file and modify the source code to suit your needs.
5. Compile the source code to generate a `class` file.

In the case of the example used in this manual, you could use this command:

```
javac -classpath c:\temp;c:\pt8\class;c:\PT8\class\psjoa.jar SDK_BUS_EXP.java
```
Understanding the Java Template

You can use the Java template as a starting point for your Java program. This section contains a skeleton of the generated Java template for a component interface named SDK_BUS_EXP, which is part of the component interface SDK. The template has been edited for length.

Import all the required classes.

```java
import java.io.*;
import psft.pt8.joa.*;
import PeopleSoft.Generated.CompIntfc.*;
public class SDK_BUS_EXP {
    public static ISession oSession;
    ......
    public static void main(String args[]) {
        try {
            //***** Set Connect Parameters *****
            String strServerName, strServerPort, strAppServerPath;
            String strUserID, strPassword;
            ......
            //Build Application Server Path
            strAppServerPath = strServerName + ":" + strServerPort;

            Note. To enable Jolt failover and load balancing in the PeopleSoft Pure Internet Architecture, you can supply multiple application server domains for the strAppServerPath variable. Separate the domain names with a comma, and make sure that no spaces are included, for example, strAppServerPath = //APPSRVR1:8000,/APPSRVR2:9000

            Create the PeopleSoft Session object to enable access to the PeopleSoft system.

            The Session object controls the environment and enables you to do error handling for all APIs from a central location.

            //***** Create PeopleSoft Session Object *****
            oSession = API.createSession();

            Connect to the Application server by using the connect method of the Session object.

            //***** Connect to the App Server *****
            if (!oSession.connect(1, strAppServerPath, strUserID, strPassword, null)) {
                System.out.println("\nUnable to Connect to Application Server.");
                ErrorHandler();
                return;
            }

            Get a reference to the component interface providing its name. (A runtime error occurs if the component interface does not exist.)
```
ISdkBusExp oSdkBusExp;
String ciName;
    ciName = "SDK_BUS_EXP";
oSdkBusExp = (ISdkBusExp) oSession.getCompIntfc(ciName);
if (oSdkBusExp == null) {
    System.out.println("Unable to Get Component Interface "+
    ciName);
    ErrorHandler();
    return;
}

//***** Set the Component Interface Mode *****
oSdkBusExp.setInteractiveMode(false);
oSdkBusExp.setGetHistoryItems(true);
oSdkBusExp.setEditHistoryItems(false);

Set the keys for the component interface. In this example, SDK_EMPLID is the Get key.

//***** Set Component Interface Get/Create Keys *****
String strSdkEmplid;
System.out.print("Enter SdkEmplid: ");
strSdkEmplid = inData.readLine();
oSdkBusExp.setSdkEmplid(strSdkEmplid);

The get() method retrieves data from the database, associated with the key values.

//***** Execute Get *****
if (!oSdkBusExp.get()) {
    System.out.println("No rows exist for the specified keys. Failed to
get the Component Interface.");
    ErrorHandler();
    return;
}

Get and print properties at level 0.

System.out.println("oSdkBusExp.SdkName: "+
oSdkBusExp.getSdkName());

Get collection at level 1 (SDK_BUS_EXP_PER).

ISdkBusExpSdkBusExpPerCollection oSdkBusExpPerCollection;
ISdkBusExpSdkBusExpPer oSdkBusExpPer;
oSdkBusExpPerCollection = oSdkBusExp.getSdkBusExpPer();

Get and print properties at level 1.

for (int i17 = 0;
i17 < oSdkBusExpPerCollection.getCount(); i17++) {
    oSdkBusExpPer = oSdkBusExpPerCollection.item(i17);

    System.out.println("oSdkBusExpPer.SdkExpPerDt: "+
oSdkBusExpPer.getSdkExpPerDt());
    ..... Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the
SDK_BUS_EXP_PER collection.
Get collection at level 2 (SDK_BUS_EXP_DTL).

```java
ISdkBusExpSdkBusExpPerSdkBusExpDtlCollection
oSdkBusExpDtlCollection;
ISdkBusExpSdkBusExpPerSdkBusExpDtl
oSdkBusExpDtl;
oSdkBusExpDtlCollection = oSdkBusExpPer.getSdkBusExpDtl();
```

Get and print properties at level 2.

```java
for (int i211 = 0;
    i211 < oSdkBusExpDtlCollection.getCount(); i211++) {
oSdkBusExpDtl = oSdkBusExpDtlCollection.item(i211);
    System.out.println("oSdkBusExpDtl.SdkChargeDt: " +
        oSdkBusExpDtl.getSdkChargeDt());
    ....
}
```

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD, SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.

```java
}
}
```

Disconnect from the Application server by using the disconnect method of the Session object. This method clears the buffers and releases the memory.

```java
//***** Disconnect from the App Server *****
oSession.disconnect();
return;
}
catch (Exception e) {
e.printStackTrace();
System.out.println("An error occurred: ");
ErrorHandler();
}
}
```
Chapter 6

Programming Component Interfaces in C++

This chapter discusses how to:

• Build the APIs for C++.
• Set up the C++ environment.
• Generate C++ runtime code templates.
• Understand the C++ template.

---

Building APIs for C++

If you plan to access your component interface from a C++ external application, you must create a component interface API. The APIs are in the form of C header files (*.h), which need to be included in the calling program.

To build the component interface bindings:

1. Open any component interface definition in PeopleSoft Application Designer.

   Use any component interface definition, because you can build APIs for all of them, regardless of which one is open.

2. Select Build, PeopleSoft APIs.

   The Build PeopleSoft API Bindings dialog box appears.

3. Select the Build check box in the C Header Files group box.

   For the target directory, enter the directory in which you want the C++ header file to be created, typically "<PS_HOME>\bin\client\winX86".

4. Click OK to build the bindings that you selected.

   The peoplesoft_peoplesoft_.i.h file that constitutes the bindings is built in the location that you specified. If the operation was successful, a Done message appears in the PeopleSoft Application Designer Build window.
Setting Up the C++ Environment

When deploying component interfaces on a local client machine with C++ bindings, you must have:

- The third-party C++ application.
- The Application server and database.
- The Java Virtual Machine (JVM) supplied with the Sun Microsystems Java Development Kit (JDK) found in the %PS_HOME%\JRE directory.
- Your compiler, configured for the C++ project.

Third-Party Applications

For applications written in C or C++, note that:

- The function names generated by the Build APIs process can be quite long. You may want to consider creating classes within your C++ code to mask this length throughout your program.
- When you create your installation for your C or C++ program, make sure that you include the setup of the path to the psapiadapter.dll.

Setting Up Client Machines to Access C++ APIs

To set up your client machine to access the component interface API using C++:

1. Install the PeopleSoft File Server.
   See PeopleSoft Installation Guide, "Using the PeopleSoft Installer."

2. Install the JVM supplied with the Sun Microsystems JDK. The JVM is located in the %PS_HOME%\JRE directory.

3. Set the environment variable PATH to include the directory containing jvm.dll.
   For example, you might set it to c:\bea\jkd<version>\jre\bin\client; or, if the PeopleTools install is done locally, the path is <PS_HOME>\jre\bin\client.

4. Set the environment variable CLASSPATH to include the psjoa.jar file (typically <PS_HOME>\class\psjoa.jar).

5. Set the environment variable PS_HOME to point to the installed PeopleSoft PeopleTools directory (for example, c:\pt849).

Configuring Compilers for C++ Projects

To configure a compiler for the C++ project:
Note. These instructions assume that you are using Microsoft Visual C++. If you use a different compiler, apply the equivalent settings for that product.

1. Create a new project in Microsoft Visual C++.
2. Select Tools, Options.
3. Select the Directories tab.
4. Click the New button in the Options dialog box.
5. Enter the path to the SDK include files, for example:
   
   `C:\PT840\SDK\PSCOMPINTFC\SRC\C++\SAMPLES\INC`

6. Click OK to save the options.
7. Open the Project Settings dialog box.
8. Select the C/C++ tab.
9. Select the General category.
10. Add PS_WIN32 to the preprocessor definitions.
11. Select the Link tab.
12. Select the Input category.
13. Specify the full path to psapiadapter.lib for the Object/library modules.

   This is typically `<PS_HOME>\src\lib\psapiadapter.lib`. Make sure that this is the only entry for `psapiadapter.lib`.
14. Click OK to save the settings.

---

Generating C++ Runtime Code Templates

To access a component interface through external APIs using C++, PeopleSoft Application Designer generates a template in the form of boilerplate C++ code that you can adapt to your purposes.

External C++ APIs are located in the `<PS_CFG_HOME>\ExtAPI_C` directory.

This section describes how to generate the template code.
Generating C++ template

To generate a C++ template for a component interface:

1. Open a component interface definition in PeopleSoft Application Designer.
2. Right-click anywhere in the definition view to display the menu.
3. Select Generate C Template.

When the template is successfully generated, a message appears stating the name and location of the template file.

**Note.** The template file is generated in the directory specified by the TEMP or TMP system environment variable on your client machine.

4. Add the generated template file to the project.
   
   In Microsoft Visual C++:
   
   a. Open the project created earlier.
   
   b. Select Project, Add To Project, Files.
   
   c. Select the generated file.
   
   d. Click OK.
5. Edit the generated file and modify the source code to suit your needs.

6. Build the project to generate an executable (.exe) file.

---

**Understanding the C++ Template**

The C++ template can be used as a starting point for your C++ program. This section contains a skeleton of the generated C++ template for a component interface named SDK_BUS_EXP, which is part of the component interface SDK. The template has been edited for length.

Include all the required header files.

```cpp
#ifdef PS_WIN32
#include "stdafx.h"
#endif

#include "cidef.h"
#include "apiadapterdef.h"
#include "PSApiAdapterInter.h"
#include "PSApiExternalLib.h"
#include "peoplesoft_peoplesoft_i.h"
#include <stdio.h>
#include <iostream.h>
#include <wchar.h>

HPSAPI_SESSION hSession;
TCHAR tmpValue[1024];

.....

void main(int argc, char* argv[])
{
    //***** Set Connect Parameters *****
    TCHAR strServerName[40], strServerPort[10], strAppServerPath[80];
    TCHAR strUserID[80], strPassword[80];
    ..... 
    //Build Application Server Path
    _stprintf(strAppServerPath, _T("%s:%s"), strServerName, strServerPort);

    Note. To enable Jolt failover and load balancing in the PeopleSoft Internet Architecture, you can supply multiple application server domains for the strAppServerPath variable. Separate the domain names with a comma, and make sure that no spaces are included, for example, strAppServerPath = //APPSRVR1:8000,//APPSRVR2:9000.

Create the PeopleSoft Session object to enable access to the PeopleSoft system.

The Session object controls the environment and enables you to perform error handling for all APIs from a central location.
//***** Create PeopleSoft Session *****
PSAPIVARBLOB ExternalAuth;
memset(&ExternalAuth, 0, sizeof(PSAPIVARBLOB));
hSession = PSApiCreateSession();
if (!hSession)
{
    wprintf(L"\nUnable to Create Session\n");
    return;
}

Connect to the Application server by using the Session_Connect() function.

//***** Connect to the App Server *****
if (!Session_Connect(hSession, 1, strAppServerPath, strUserID, strPassword," ExternalAuth))
{
    wprintf(L"\nUnable to Connect to Application Server\n");
    ErrorHandler();
    return;
}

Get a reference to the component interface providing its name. (A runtime error occurs if the component interface does not exist.)

//***** Get Component Interface *****
HPSAPI_SDK_BUS_EXP hSdkBusExp;
TCHAR ciName[30];
_tscscpy(ciName, _T("SDK_BUS_EXP"));
hSdkBusExp = (HPSAPI_SDK_BUS_EXP) Session_GetCompIntfc(hSession, ciName);
if (!hSdkBusExp)
{
    wprintf(L"\nUnable to Get Component Interface %s\n", ciName);
    ErrorHandler();
    return;
}

//***** Set the Component Interface Mode *****
SdkBusExp_SetInteractiveMode(hSdkBusExp, false);
SdkBusExp_SetGetHistoryItems(hSdkBusExp, true);
SdkBusExp_SetEditHistoryItems(hSdkBusExp, false);

Set the keys for the component interface. In this example, SDK_EMPLID is the Get key.

//***** Set Component Interface Get/Create Keys *****
TCHAR strSdkEmplid[80];
wprintf(L"\nEnter SdkEmplid: ");
_getts(strSdkEmplid);
SdkBusExp_SetSdkEmplid(hSdkBusExp, strSdkEmplid);

The <CI_NAME>_Get() function retrieves data from the database associated with the key values.

//***** Execute Get *****
if (!SdkBusExp_Get(hSdkBusExp))
{
    wprintf(L"\nUnable to Get Component for the Search keys provided.\n");
    ErrorHandler();
    return;
}

Get and print properties at level 0.
Similar code is generated for the properties SDK_BIRTHDATE and SDK_DEPTID.

Get collection at level 1 (SDK_BUS_EXP_PER).

```c
HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PERCOLLECTION
hSdkBusExpSdkBusExpPerCollection;
HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PER hSdkBusExpSdkBusExpPer;
hSdkBusExpSdkBusExpPerCollection = SdkBusExp_GetSdkBusExpPer(hSdkBusExp);
Get and print properties at level 1.

for (int i17 = 0; i17 < SdkBusExpSdkBusExpPerCollection_GetCount (hSdkBusExpSdkBusExpPerCollection); i17++)
{
    hSdkBusExpSdkBusExpPer = SdkBusExpSdkBusExpPerCollection_Item (hSdkBusExpSdkBusExpPerCollection, i17);
    wprintf(L"oSdkBusExpSdkBusExpPer.SdkExpPerDt: %s\n",
            printProperty (SdkBusExpSdkBusExpPer_GetSdkExpPerDt(hSdkBusExpSdkBusExpPer), tmpValue));
}
```

Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the SDK_BUS_EXP_PER collection.

Get collection at level 2 (SDK_BUS_EXP_DTL).

```c
HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PER_SDK_BUS_EXP_DTLCOLLECTION
hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection;
HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PER_SDK_BUS_EXP_DTL hSdkBusExpSdkBusExpPerSdkBusExpDtl;
hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection = SdkBusExpSdkBusExpPer_GetSdkBusExpDtl(hSdkBusExpSdkBusExpPer);
Get and print properties at level 2.

for (int i211 = 0; i211 < SdkBusExpSdkBusExpPerSdkBusExpDtlCollection_GetCount (hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection); i211++)
{
    hSdkBusExpSdkBusExpPerSdkBusExpDtl =
        SdkBusExpSdkBusExpPerSdkBusExpDtlCollection_Item (hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection, i211);
    wprintf(L"oSdkBusExpSdkBusExpPerSdkBusExpDtl.SdkChargeDt: %s\n",
            printProperty (SdkBusExpSdkBusExpPerSdkBusExpDtl_GetSdkChargeDt (hSdkBusExpSdkBusExpPerSdkBusExpDtl), tmpValue));
}
```

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD, SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.

Disconnect from the Application server by using the disconnect method of the Session object. This method clears the buffers and releases the memory.
//***** Disconnect from the App Server *****
Session_Disconnect(hSession);
return;
}
Chapter 7

Programming Component Interfaces in COM

This chapter discusses how to:

- Build APIs for COM.
- Set up the COM environment.
- Generate a Visual Basic runtime code template.
- Understand the Visual Basic template.

Building APIs for COM

If you plan to access your component interface from a COM external application, you must create a component interface API. The generated APIs are in the form of registry entries and type library files.

To build the component interface bindings:

1. Open any component interface definition in PeopleSoft Application Designer.
   
   Use any component interface definition, because you can build APIs for all of them, regardless of which one is open.

2. Select Build, PeopleSoft APIs.
   
   The Build PeopleSoft API Bindings dialog box appears.

3. Select the Build check box in the COM Type Library group box.
   
   a. For the target directory, enter the directory in which you want the COM type library to be created, typically `<PS_HOME>in\client\winX86`.

   b. Enter the COM server DLL location to specify where the PeopleSoft API Adapter (psapiadapter.dll) is typically located: `<PS_HOME>in\client\winX86`.

4. (Optional) Select the AutoRegister check box to execute the registry file immediately upon building the API.
   
   This causes your client machine registry to be updated immediately without having to register the registry entry manually.
5. (Optional) Select the Clean-up Registry check box to clean up the registry if you previously generated the Peoplesoft_Peoplesoft.reg file.

This is needed so that the older registry settings do not remain and conflict with settings made by the latest version.

6. Click OK to build the bindings that you selected.

The files that constitute the bindings are built in the location that you specified. If the operation was successful, a Done message appears in the PeopleSoft Application Designer Build window and the client machine should contain a Peoplesoft_Peoplesoft.reg and PeopleSoft_PeopleSoft.tlb file.

---

### Setting Up the COM Environment

When deploying component interfaces on a local client machine or web server with COM bindings, you need to have:

- The third-party COM application.
- The application server and database.
- The Java Virtual Machine (JVM) supplied the with Sun Microsystems Java Development Kit (JDK), found in the %PS_HOME%\JRE directory.
- A copy of the type library called PeopleSoft_PeopleSoft.tlb that you generated during the Build PeopleSoft API Bindings process.

  This type library is not specific to a single database instance—it is specific to those database objects.

- A copy of the registry file called PeopleSoft_PeopleSoft.reg that you generated during the Build PeopleSoft API Bindings process.

  This registry file is not specific to a single database instance.

To set up your client machine to access the component interface API, using COM:

**Note.** A reboot of the client machine may be required after you follow these steps.

1. Install the PeopleSoft file server.

   See *PeopleSoft 8.50 Installation Guide, "Using the PeopleSoft Installer."

2. Set the environment variable PATH to include the directory containing jvm.dll.

   For example, you might set it to c:\bea\jkd<version>\jre\bin\client; or, if the PeopleTools installation is done locally, the path is <PS_HOME>\jre\bin\client.

3. Set the environment variable CLASSPATH to include the file psjoa.jar (typically <PS_HOME>\class\psjoa.jar).

**Note.** The following steps assume that you are using Microsoft Visual Basic. If you use a different compiler, apply the equivalent settings for that product.
4. Open the Visual Project File pscitester.vbp or sdk_bus_exp.vbp in Microsoft Visual Basic.

5. Select Project, References, and add the Peoplesoft_PeopleSoft.tlb type library.

**Third-Party Application**

Copy the type library and registry files to the directory containing the external API on each client machine from which you want to use the COM API.

Apply the API registry settings by double-clicking Peoplesoft_PeopleSoft.reg.

---

**Warning!** The registry file includes references to the external API and type library files and their locations on the original client machine where they were built, so those files must be in the same locations on the current client. If the directory structure is different, you must edit Peoplesoft_PeopleSoft.reg to reflect the current machine before you apply the registry settings.

- If your program is early-binding, the code contains a direct reference to the path of the type library. Therefore, as you deploy, you must have the type library in the same directory on each machine.
- If your program is late-binding, the code does not contain a reference to the path of the type library. The code looks in the registry for the path to the type library. Therefore, as you deploy, you can have the type library in different directories on each machine. You must update the registry settings as part of the deployment. This is a more flexible approach.

---

**Generating Visual Basic Runtime Code Templates**

When you want to access a component interface through external APIs by using Microsoft Visual Basic, PeopleSoft Application Designer generates a template in the form of boilerplate Visual Basic code that you can adapt to your purposes.

External COM APIs are located in the `<PS_CFG_HOME>\ExtAPI_COM` directory.

This section describes how to generate the template code.
Generating Visual Basic template

To generate a Visual Basic template for your component interface:

1. Open the desired component interface definition in PeopleSoft Application Designer.

2. Right-click anywhere in the definition view to display the menu.


   When the template is successfully generated, a message appears stating the name and location of the template file.

   **Note.** The template file is generated in the directory specified by the TEMP or TMP system environment variable on your client machine.

4. Open the generated file and modify the source code as needed.

---

**Understanding Visual Basic Templates**

You can use the Visual Basic template as a starting point for your Visual Basic program. This section contains a skeleton of the generated Visual Basic template for a component interface named SDK_BUS_EXP, which is part of the component interface SDK. The template has been edited for length.
Declare the Session object.

Dim oSession As PeopleSoft_PeopleSoft.Session
.....

Private Sub main()
  On Error GoTo ErrorHandler
  '***** Set Connect Parameters *****
  Dim strServerName As String, strServerPort As String,
  strAppServerPath As String
  Dim strUserID As String, strPassword As String
  ......

  'Build Application Server Path
  strAppServerPath = strServerName & ":" & strServerPort

  Note. To enable Jolt failover and load balancing in the PeopleSoft Internet Architecture, you can supply multiple application server domains for the strAppServerPath variable. Separate the domain names with a comma, and make sure no spaces are included, for example:

  strAppServerPath = //APPSRVR1:8000,//APPSRVR2:9000

Create the PeopleSoft Session object to enable access to the PeopleSoft system. The Session object controls the environment and enables you to do error handling for all APIs from a central location.

  '***** Create PeopleSoft Session Object *****
  Set oSession = CreateObject("PeopleSoft.Session")

Connect to the Application server by using the Connect method.

  '***** Connect to the App Server *****
  If Not oSession.Connect(1, strAppServerPath, strUserID, strPassword, 0) Then
    Err.Raise 1001, "", "Unable to connect to Application Server"
    Call ErrorHandler()
    Exit Sub
  End If

Get a reference to the component interface providing its name. A runtime error occurs if the component interface does not exist.

  Dim oSdkBusExp As SDK_BUS_EXP
  Dim ciName As String
  ciName = "SDK_BUS_EXP"
  Set oSdkBusExp = oSession.GetCompIntfc(ciName)
  If oSdkBusExp Is Nothing Then
    Err.Raise 1001, "", "Unable to Get Component Interface " & ciName
    Call ErrorHandler()
    Exit Sub
  End If

  '***** Set the Component Interface Mode *****
  oSdkBusExp.InteractiveMode = False
  oSdkBusExp.GetHistoryItems = True
  oSdkBusExp.EditHistoryItems = False

Set the keys for the component interface. In this example, SDK_EMPLID is the Get key.

  '***** Set Component Interface Get/Create Keys *****
  Dim strSDK_EMPLID As String
  strSDK_EMPLID = InputBox("Enter SDK_EMPLID: ")
  oSdkBusExp.SDK_EMPLID = strSDK_EMPLID
The Get method retrieves data from the database associated with the key values.

```vbnet
'***** Execute Get *****
If Not oSdkBusExp.Get() Then
    Err.Raise 1001, ",", ", , No rows exist for the specified keys. Failed to get the" Component Interface"
    Call ErrorHandler()
    Exit Sub
End If
```

Get and print properties at level 0.

```vbnet
Debug.Print "oSdkBusExp.SDK_NAME: " & oSdkBusExp.SDK_NAME
oSdkBusExp.SDK_NAME = <*>
```

Similar code is generated for the properties SDK_BIRTHDATE and SDK_DEPTID.

Get the collection at level 1 (SDK_BUS_EXP_PER).

```vbnet
Dim oSdkBusExpPerCollection As
   SDK_BUS_EXP_SDK_BUS_EXP_PERCollection
Dim oSdkBusExpPer As SDK_BUS_EXP_SDK_BUS_EXP_PER
Set oSdkBusExpPerCollection = oSdkBusExp.SDK_BUS_EXP_PER
```

Get and print properties at level 1.

```vbnet
Dim i17 As Integer
For i17 = 1 To oSdkBusExpPerCollection.Count
    Set oSdkBusExpPer = oSdkBusExpPerCollection.Item(i17)
    Debug.Print "oSdkBusExpPer.SDK_EXP_PER_DT: " & 
        oSdkBusExpPer.SDK_EXP_PER_DT
```

Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the
SDK_BUS_EXP_PER collection.

Get collection at level 2 (SDK_BUS_EXP_DTL).

```vbnet
Dim oSdkBusExpDtlCollection As
   SDK_BUS_EXP_SDK_BUS_EXP_PER_SDK_BUS_EXP_DTLCollection
Dim oSdkBusExpDtl As SDK_BUS_EXP_SDK_BUS_EXP_PER_SDK_BUS_EXP_DTL
Set oSdkBusExpDtlCollection = oSdkBusExpPer.SDK_BUS_EXP_DTL
```

Get and print properties at level 2.

```vbnet
Dim i211 As Integer
For i211 = 1 to oSdkBusExpDtlCollection.Count
    Set oSdkBusExpDtl = oSdkBusExpDtlCollection.Item(i211)
    Debug.Print "oSdkBusExpDtl.SDK_CHARGE_DT: " & 
        oSdkBusExpDtl.SDK_CHARGE_DT
```

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD,
SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.

```vbnet
Next
Next
```

Disconnect from the Application server by using the disconnect method of the Session object. This method clears the buffers and releases the memory.
***** Disconnect from the App Server *****
If Not oSession Is Nothing Then
  oSession.Disconnect
  Set oSession = Nothing
End If
Exit Sub
.....
End Sub
Chapter 8

Using the Component Interface Software Development Kit

This chapter provides an overview of the component interface software development kit (SDK) and discusses how to:

- Set SDK prerequisites.
- Use the SDK_BUS_EXPENSES test page.
- Test the SDK_BUS_EXP component interface.
- Use the component interface SDK Java and C++ sample.
- Use the component interface SDK COM Excel sample.
- Use the component interface SDK COM ASP sample.

Understanding the Component Interface SDK

The PeopleSoft component interface SDK is installed with the PeopleTools installation. It provides resources to assist you in developing and testing component interface-based integration between PeopleSoft and third-party applications. The SDK contains sample definitions with data and source code. For easy identification, all of the definition names start with SDK_. The SDK is installed in the PeopleSoft home directory (PS_HOME) under sdk.

**Note.** The SDK definitions and associated data are for development purposes only and should not be used in a production environment.

Component Interface SDK Samples

Programming samples for the component interface SDK_BUS_EXP are part of the SDK. The samples are available in four languages—Java, C++, VBA, and ASP.

The component interface source code is located in the <PS_HOME>\SDK\PSCOMPINTFC directory.

**Note.** The source files mentioned in this section are located relative to the installed PeopleSoft home directory (PS_HOME).
Prerequisites for Using the Component Interface SDK

To call a PeopleSoft component interface, you must have:

- A working understanding of PeopleTools components.
- A working understanding of Java, C++, or COM.
- The application server and database installed.
- The Java Virtual Machine (JVM) installed that is supplied with the Sun Microsystems Java Development Kit (JDK), found in the %PS_HOME%/JRE directory.

Using the SDK_BUS_EXPENSES Test Page

The SDK includes a component interface, called SDK_BUS_EXP, which is part of the sample development project and is delivered with the SDK. It is built on the component SDK_CISAMPLES, which contains the page SDK_BUS_EXP. The page exposes information about employee business expenses for external access.

Note. The component SDK_CISAMPLES is a sample and is not for business use.

SDK_BUS_EXPENSES page

To test the SDK_BUS_EXPENSES test page:

1. Provide access to the SDK_CISAMPLES component, using PeopleTools security.
2. Select PeopleTools SDK, PeopleTools SDK, Use, SDK CI Samples.
3. Search for and select an employee ID.

---

**Testing the SDK_BUS_EXP Component Interface**

To test the SDK_BUS_EXP component interface:

1. View the component interface definition through the PeopleSoft Application Designer.
2. Test the component interface definition, using the component interface tester.

---

**Using the Component Interface SDK Sample in Java and C++**

This section describes how to use the component interface SDK sample in Java and C++.

**Understanding using the Component Interface SDK Samples in Java and C++**

The component interface sample programs for Java and for C++ are provided as part of the component interface SDK and follow the same sequence of options. The source files are located in `<PS_HOME>/sdk/pscompintfc/src<java or c++>/samples/sdk_bus_exp`.

**Building the Component Interface SDK Sample (Java)**

The component interface sample program for Java is provided as part of the component interface SDK, located in `<PS_HOME>/sdk/pscompintfc/src/java/samples/sdk_bus_exp`.

The Java source code for the sample is in the following file: `sdk_bus_exp.java`

Before you run the sample, you must build the APIs and set up the Java environment.

To build the Java component interface sample:

1. Set your java classpath to include the external API classes you already built and the psoa.jar library delivered under `<PS_HOME>/class/psjoa.jar`
2. Compile the source using `javac sdk_bus_exp.java`

**Building the Component Interface Sample (C++)**

The component interface sample program for C/C++ is provided as part of the component interface SDK, located in `<PS_HOME>/sdk/pscompintfc/src/cpp/samples/sdk_bus_exp`.

The C++ source code for the sample is in the following file: `sdk_bus_exp\sdk_bus_exp.cpp`
Before you run the sample, you must build the APIs and set up the C++ environment. To build the C++ component interface sample:

1. Open the sdk_bus_exp workspace in the Microsoft Visual C++ editor.
2. Build the project by selecting Build, Rebuild All.

Running the Component Interface SDK Sample in Java and C++

To run the compiled Java or C++ component interface sample:

1. In a DOS window, change directories to the location of the sdk_bus_exp directory.

   After you launch the executable sdk_bus_exp, the system prompts you for parameters one at a time.

2. At each prompt, enter the appropriate value and press Enter.

   Select option 1 to sign in. You are then prompted to provide the connect information.

   If the connect succeeds, a menu appears where you can perform Get or Find functions.

3. Get details for an employee.

   Select option 1 to get details for an employee. You are then prompted with the different update modes and the employee ID for which you want to display information. Enter the employee ID 8001 and press Enter. This displays the level 0 data and the options that you can perform.

4. Select a business expense period at collection level 1.

   Select option 8, Item, to select a business expense period. Selecting this option displays a list of available business expense periods for the selected employee.

   Select the expense period that you want to work with.

5. Select a business expense detail item at collection level 2.

   Select option 18, Item, to select a business expense detail within the selected business expense period. Selecting this option displays a list of available business expense detail items within the selected business expense periods.

Interpreting the Code for the Component Interface SDK Sample (Java)

The following discussion refers to the Java sample program, sdk_bus_exp.java. (The code has been edited for length.) It explains the runtime options shown above.
1. Import all the required classes:

The code example shows how to import the required classes:

```java
package sdk_bus_exp;
import java.io.*;
import java.util.*;
import psft.pt8.joa.*;
import PeopleSoft.Generated.CompIntfc.*;
public class sdk_bus_exp {
    ...
```

2. Declare all the required objects.

Only one active period and one active detail record are possible at any time. Users are prompted to select
the needed values if they are not active.

<table>
<thead>
<tr>
<th>Collection Object</th>
<th>Level</th>
<th>Item Object for Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSdkBusExpCollection</td>
<td>Root (SDK_BUS_EXP)</td>
<td>oSdkBusExp</td>
</tr>
<tr>
<td>oSdkBusExpPerCollection</td>
<td>Level 1 (SDK_BUS_EXP_PER)</td>
<td>oSdkBusExpPer</td>
</tr>
<tr>
<td>oSdkBusExpDtlCollection</td>
<td>Level 2 (SDK_BUS_EXP_PER_DTL)</td>
<td>oSdkBusExpDtl</td>
</tr>
</tbody>
</table>

In addition, the CompIntfPropInfoCollection object is used to access the structure of a component
interface. It is not specific to a component interface.

3. Declare the PeopleSoft session object.

4. Connect to the application server.

5. Instantiate the component interface.

6. Perform a Get or Create to access the component interface.

You must provide the keys to access the record that you want to modify.

7. Use the appropriate methods to access the component interface properties.

There are standard methods and user-defined methods defined for the session, the component interface,
and the component interface collections.

The executeMethod function is used to invoke a method specified as a function parameter (nMethodIn).

The component interface Java SDK sample has 25 options:

<table>
<thead>
<tr>
<th>SDK Option</th>
<th>Where Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 through 5</td>
<td>On the component interface.</td>
</tr>
<tr>
<td>6 through 15</td>
<td>SDK_BUS_EXP_PER collection.</td>
</tr>
<tr>
<td>16 through 25</td>
<td>SDK_BUS_EXP_DTL collection</td>
</tr>
</tbody>
</table>
Options 1 through 4 and options 6 through 25 are similar in behavior to those described in PeopleCode API PeopleBook for a component Interface and its collections.

Option 5, InsertBusExpDtlDefaults, is the user-defined method of the SDK_BUS_EXP component interface. This method is defined in PeopleCode inside the component interface definition.

The logic used in the corresponding options of these collections is identical.

See Enterprise PeopleTools 8.50 PeopleBook: PeopleCode API Reference, "Component Interface Classes."

This is the main method. It performs such functions as starting the session, getting the component interface, and disconnecting:

```java
public static final void main(String[] args) {
    System.out.println("  ");
    System.out.println(" 1) Sign In  ");
    System.out.println("  q) Quit ");
    System.out.println("  ");
    System.out.print("Command to execute (1, q) [1]: ");
    char Temp = readCharacter();
    switch (char Temp) {
        case 'q': case 'Q':
            disconnectFromAppServer();
            return;
        default:
            getConnectParameters();
            if (connectToAppServer()) {
                oSdkBusExp = (ISdkBusExp) oSession.getCompIntfc(m_strCIName);
                while (getKeyType()) {
                    methodInt = selectMethod();
                    while (methodInt != 0) {
                        executeMethod(methodInt);
                    }
                }
            }
            methodInt = 0;
            methodInt = selectMethod();
            ......
```
3. Declare all the required objects. Only one active period and one active detail record are possible at any time.

<table>
<thead>
<tr>
<th>Collection Object</th>
<th>Level</th>
<th>Item Object for Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>hSdkBusExpCollection</td>
<td>Root (SDK_BUS_EXP)</td>
<td>hSdkBusExp</td>
</tr>
<tr>
<td>hSdkBusExpPerCollection</td>
<td>Level 1 (SDK_BUS_EXP_PER)</td>
<td>hSdkBusExpPer</td>
</tr>
<tr>
<td>hSdkBusExpDtlCollection</td>
<td>Level 2 (SDK_BUS_EXP_PER_DTL)</td>
<td>hSdkBusExpDtl</td>
</tr>
</tbody>
</table>

The function `executeMethod` is used to launch the appropriate method depending upon the user input (`nMethodIn`).

The component interface C++ SDK sample has 25 options:

SDK Option 1 through 5 SDK_BUS_EXP_PER collection, 6 through 25 SDK_BUS_EXP_DTL collection

<table>
<thead>
<tr>
<th>SDK Option</th>
<th>Where Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 through 5</td>
<td>On the component interface.</td>
</tr>
<tr>
<td>6 through 15</td>
<td>SDK_BUS_EXP_PER collection</td>
</tr>
<tr>
<td>16 through 25</td>
<td>SDK_BUS_EXP_DTL collection</td>
</tr>
</tbody>
</table>

Options 1 through 4 and options 6 through 25 are similar in behavior to those described in PeopleCode API PeopleBook for a component Interface and its collections.

Option 5, `InsertBusExpDtlDefaults`, is the user-defined method of the SDK_BUS_EXP component interface. This method is defined in peoplecode inside the component interface definition.

The logic used in the corresponding options of these collections is identical.

See *Enterprise PeopleTools 8.50 PeopleBook: PeopleCode API Reference*, "Component Interface Classes."

**Using the Component Interface SDK COM Excel Sample**

The component interface sample program for Microsoft Excel is provided as part of the component interface SDK, located in `<PS_HOME>\sdk\pscompintfc\src\com\samples\vba`.

The Visual Basic source code for the sample is in the following file: `sdk_bus_exp.xls`

Before you run the sample, you must build the APIs and set up the COM environment.
Running the Component Interface SDK COM Excel Sample

When running the Microsoft Excel sample, you use the Get and Find sheet to find an employee.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Empl Id</strong></td>
<td><strong>80</strong></td>
</tr>
<tr>
<td><strong>Last Name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Enter full Employee ID for Get function</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Enter partial or no information for Find function</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Select an Employee ID, and press Get selected after search</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Business expense Period for Schumacher,Simon empid 8001</strong></td>
<td></td>
</tr>
<tr>
<td><strong>8001 ASD</strong></td>
<td>Schumacher,Simon</td>
</tr>
<tr>
<td><strong>8052 AVERY</strong></td>
<td>Avery,Joan</td>
</tr>
</tbody>
</table>

Sheet 2: Find and Get an Employee

To run the Microsoft Excel component interface sample:

1. Launch Microsoft Excel.
2. Open the Microsoft Excel sample spreadsheet.
   The Microsoft Excel spreadsheet is located in `<PS_HOME>\sdk\pscompintfc\src\com\samples\vba\sdk_bus_exp`.
3. When prompted about macros, select Enable Macros.
4. Attach PeopleSoft References to the spreadsheet.
   This example uses early bindings and hence requires attaching references to the spreadsheet. Select Tools, Macro, Visual Basic Editor from the Microsoft Excel menu. This opens the VBA editor.
   Select Tools, References from the menu. A dialog box appears, listing all the available references. Select the reference PeopleSoft_PeopleSoft.
5. Sign in to the sdk_bus_exp sample.
   Sheet 1 of the sdk_bus_exp spreadsheet is the sign-in page. Provide the connect information and press Tab to navigate out of the fields. Click Connect to establish the connection.
6. Find an employee by using the Find keys.
   The Find and Get keys are located on Sheet 2.
7. Select an employee from the list.
   Select an employee ID from the list by making the cell active and then clicking the GET selected button.
8. Get an Employee by providing the Get key.

   Enter the complete employee ID in cell B3. Press Tab to navigate out of the cell, and click Get (EMPLID). A list of all the available periods is displayed.


   To view the details for the listed business expense periods, click the Toggle Details button.

10. Add a new business expense period.

   Click the Insert period button. This redirects you to Sheet 3. Enter the business expense period date. Press Tab and click the Save New Period button.

11. Add a new business expense detail.

   Click the Insert Detail button. This redirects you to Sheet 3. Enter the charge date, expense code, amount, department ID, and business purpose. Press Tab, and save the new detail by clicking the Save New Detail button.

You can list the expense periods for the employee.

<table>
<thead>
<tr>
<th></th>
<th>Business expense Period for Schumacher, Simon emplid 8001</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Period Number</td>
</tr>
<tr>
<td>12</td>
<td>Period # 1</td>
</tr>
<tr>
<td>13</td>
<td>Period # 2</td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Expense periods

Understanding the Component Interface SDK COM Excel Sample Code

The following listings of code are taken from the Microsoft Excel sample program, sdk_bus_exp.xls. (The code has been edited for length.)

View the code by selecting Tools, Macro, Visual Basic Editor from the menu.

- Declare the PeopleSoft session object.

- List Business Expense Periods, using the Item method to get a specific item of the type SDK_BUS_EXP_PER.

- List Business Expense Details, using the Item method to get a specific item of the type SDK_BUS_EXP_DTL.

- To save a new business expense period, use the InsertItem method. This method inserts a new row and returns an item of the type SDK_BUS_EXP_PER. The item contains the properties. Set the properties and execute the Save method.

- To save a new business expense detail, use the InsertItem method. This method inserts a new row and returns an item of the type SDK_BUS_EXP_DTL. The item contains the properties. Set the properties and execute the Save method.
Using the Component Interface SDK COM ASP Sample

The component interface sample program for ASP is provided as part of the component interface SDK, located in `<PS_HOME>/sdk/pscompintfc/src/com/samples/asp/sdk_bus_exp`.

The ASP source code for the samples is in these files.

Before you run the samples, you must build the APIs and set up the COM environment.

Running the Component Interface SDK COM ASP Sample

When running the ASP sample, you use the Get key to find an employee.

**SDK Business Expenses Component Interface - Search**

[Sign Off]

*Get* Keys for Component Interface

<table>
<thead>
<tr>
<th>SDK_EMPLID</th>
<th>8001</th>
</tr>
</thead>
</table>

ASP Get key

To run the Component Interface SDK COM ASP component interface sample:

1. Install and configure the IIS web server.
2. Create a virtual directory to point to `<PS_HOME>/sdk/pscompintfc/src/com/samples/asp/sdk_bus_exp`.
3. Start the web server.
4. Run the SDK example through the browser.

   The web address `http://machinename/sdkSDK_BUS_EXP_Signon.asp` launches the SDK application.

5. Provide the connect information and click Submit.


   Enter the Get key (SDK_EMPLID) and click the Get button. This lists all the business periods for the selected employee ID.

7. Update a business expense period.
   a. Click the Update button to update the business expense period.
   b. Update the expense period end date.
   c. Click the Save button.
8. Insert a business expense period.
   a. Click the Insert button to update the business expense period.
   b. Add the new expense period end date.
   c. Click the Save button.

9. Delete a business expense period.
   a. Click the Delete button to delete the business expense period. You are prompted to decide to delete the row.
   b. Click OK to confirm the delete.
   c. Click Cancel to cancel the operation

10. Update a business expense detail.
    Select a business expense period by clicking the Update button from the business expense period row.

11. Insert a business expense detail.
    a. Click the Insert button to insert a new business expense period.
    b. Enter the values for charge date, expense code, expense amount, currency code, business purpose, and department ID.
    c. Click the Save button to save changes.

12. Delete a business expense detail.
    a. Click the Delete button to delete the business expense detail.
       You are prompted to decide to delete the row.
    b. Click OK to confirm the delete.
    c. Click Cancel to cancel the operation.

You can list the expense periods for the employee:

**Listing Details for Employee ID: 8001**

<table>
<thead>
<tr>
<th>Employee Number</th>
<th>Period Number</th>
<th>Period Total</th>
<th>Period End Date</th>
<th>Work with Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>8001</td>
<td>1</td>
<td>1660.91</td>
<td>05/16/2001</td>
<td>Update</td>
</tr>
<tr>
<td>8001</td>
<td>2</td>
<td>223.95</td>
<td>11/09/2000</td>
<td>Update</td>
</tr>
</tbody>
</table>

Next

Expense periods
Understanding the Component Interface SDK COM ASP Sample Code

This section discusses the ASP files that are included in the component interface SDK COM ASP. The files included are described in the following table:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK_BUS_EXP_DeleteBusinessExpenseDetail.asp</td>
<td>Deletes a specific expense detail row an allows to insert or delete expense details from existing expense periods.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_DeleteBusinessExpensePeriod.asp</td>
<td>Deletes a specific expense period an allows to insert or delete expense details from existing expense periods.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_FUNCLIB.asp</td>
<td>Contains all the common functions and needs to be included in any other ASP page when you build the ASP program</td>
</tr>
<tr>
<td>SDK_BUS_EXP_GetBusinessExpenses.asp</td>
<td>Lists all the business expense periods for the selected employee.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_GetSearchParameters.asp</td>
<td>Prompts for the Find and Get Keys.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_GetSearchResults.asp</td>
<td>Lists all the employees for the provided Find keys</td>
</tr>
<tr>
<td>SDK_BUS_EXP_InsertBusinessExpenseDetail.asp</td>
<td>Updates the expense period data, as well as inserts and deletes business expense data.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_InsertBusinessExpensePeriod.asp</td>
<td>Inserts a new business expense period.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_NewBusinessExpensePeriod.asp</td>
<td>Inserts a new expense period and end date, provides options to insert or delete expense details.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_SaveBusinessExpenseDetail.asp</td>
<td>Saves submitted business expense details data.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_SaveBusinessExpensePeriod.asp</td>
<td>Updates the end date of an existing expense period, and provides options to insert or delete expense details.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_Signon.asp</td>
<td>Provides signon information and connects to the application server.</td>
</tr>
<tr>
<td>SDK_BUS_EXP_UpdateBusinessExpensePeriod.asp</td>
<td>Updates or deletes expense period details data from and existing expense period.</td>
</tr>
</tbody>
</table>

**SDK_BUS_EXP_DeleteBusinessExpenseDetail.asp**

This ASP file deletes a specific expense detail row an allows to insert or delete expense details from existing expense periods.

To use the SDK_BUS_EXP_DeleteBusinessExpenseDetail.asp page:

1. Get the connection information forwarded from the previous page.
2. Get the key fields.
4. Get the SDK_BUS_EXP_PER collection, using the Item method.
5. Execute the DeleteItem method.
6. Execute the Save method.

**SDK_BUS_EXP_DeleteBusinessExpensePeriod.asp**

This ASP file deletes a specific expense period and allows to insert or delete expense details from existing expense periods.

To use the SDK_BUS_EXP_DeleteBusinessExpensePeriod.asp page:
1. Get the connection information forwarded from the previous page.
2. Get the key fields.
3. The submitted expense period is deleted.
   
   To view the updated SDK_BUS_EXP_PER collection, execute the Cancel method. Set the keys and execute the Get method. The function getBusinessExpensePeriods displays the business expense periods.
4. Execute the Save method.
5. Execute the Cancel method.

**SDK_BUS_EXP_FUNCLIB.asp**

This ASP file contains all the common functions and needs to be included in any other ASP page when you build the ASP program. It includes utility functions for checking required fields, extracting the host name to which the client connects, and confirming and submitting the user input.

**SDK_BUS_EXP_GetBusinessExpenses.asp**

This ASP file lists all the business expense periods for the selected employee.

To use the SDK_BUS_EXP_GetBusinessExpenses.asp page:
1. Get the connection information forwarded from the previous page.
2. Get the Key Field.
3. Set the Component Interface Get Key.
4. Get the business expense periods by executing the function getBusinessExpensePeriods.
   
   The function getBusinessExpensePeriods gets the business expense period, and then loops through the collection, using the Item method to get a specific business expense period. Each property in that item is then displayed.
5. The Update button is of the type submit.

Because the form action is set to SDK_BUS_EXP_UpdateBusinessExpensePeriod.asp, this page is launched. The Insert and Delete buttons use the JavaScript functions insertBusinessExpensePeriod() and deleteBusinessExpensePeriod().

6. Submit the page.

Use the JavaScript function insertBusinessExpensePeriod to set the form.action to SDK_BUS_EXP_InsertBusinessExpensePeriod.asp and submit the page.

**SDK_BUS_EXP_GetSearchParameters.asp**

This ASP file prompts for the Find and Get Keys. You can also set the component interface modes: Interactive Mode, Get History Items, and Edit History Items.

To use the SDK_BUS_EXP_GetSearchParameters.asp page:

1. Get the connection information forwarded from the previous page.
2. Use Sub getSearchParameters prompts the user for the Get or Find key and the component interface modes (interactive, get history items, and edit history items).
3. Use Sub getSearchParameters to call the appropriate page for Get and Find, using the JavaScript function invokeMethod().

**SDK_BUS_EXP_GetSearchResults.asp**

This ASP file is called if the Find option was selected.

To use the SDK_BUS_EXP_GetSearchResults.asp page:

1. Get the connection information forwarded from the previous page.
2. Get the Find keys.
3. Get the search result.

   The function getSearchResults lists all the employees for the provided Find keys by setting the Find keys and executing the Find method.

4. Loop through the collection to list all the employee IDs.

**SDK_BUS_EXP_InsertBusinessExpenseDetail.asp**

This ASP file enables the user to update the expense period data, as well as insert and delete business expense data. Sub insertBusinessExpenseDetail inserts a business expense detail for the selected business expense period.

To use the SDK_BUS_EXP_InsertBusinessExpenseDetail.asp page:

1. Get the SDK_BUS_EXP_PER collection in Sub insertBusinessExpenseDetail.
2. Get the SDK_BUS_EXP_PER collection, using the Item method.
3. Get the SDK_BUS_EXP_DTL collection.
4. Get the SDK_BUS_EXP_DTL using the Item method.

5. Create a form to get the properties for SDK_BUS_EXP_DTL.

6. Get the connection information forwarded from the previous page.

7. Get the key fields.

8. Submit the form.

   The Save button calls the JavaScript function saveBusinessExpenseDetail, sets action of the form to SDK_BUS_EXP_SaveBusinessExpenseDetail.asp, and submits the form.

**SDK_BUS_EXP_InsertBusinessExpensePeriod.asp**

This ASP file enables the user to insert a new business expense period.

To use the SDK_BUS_EXP_InsertBusinessExpensePeriod.asp page:

1. Get the connection information forwarded from the previous page.

2. Get the key fields.

3. Call the insertBusinessExpensePeriod function to insert a new business expense period.

4. Get the business expense period.

5. Insert a new item into the collection.

   Use the InsertItem method is used to insert a new item in the SDK_BUS_EXP_PER collection.

   A field to enter the SDK_BUS_PER_DT is created.

6. Submit the form.

   Use the Save button to the JavaScript function newBusinessExpensePeriod. This function sets the action of the form to SDK_BUS_EXP_NewBusinessExpensePeriod.asp and submits the form.

**SDK_BUS_EXP_NewBusinessExpensePeriod.asp**

This ASP file enables the user to update the expense period data as well as insert and delete business expense details.

To use the SDK_BUS_EXP_NewBusinessExpensePeriod.asp page:

1. Get the connection information forwarded from the previous page.

2. Get the key fields.

3. Get the SDK_BUS_EXP_PER_COLLECTION.

   Execute the InsertItem method. Set the SDK_EXP_PER_DT property and execute the Save method.

**SDK_BUS_EXP_SaveBusinessExpenseDetail.asp**

This ASP file enables the user save submitted business expense details.
To use the SDK_BUS_EXP_SaveBusinessExpenseDetail.asp page:

1. Get the connection information forwarded from the previous page.

2. Get the key fields.

3. Save the business expense detail and get the SDK_BUS_EXP_PER collection.
   
   Using Submit saveBusinessExpenseDetails accomplishes both of these tasks. First it saves the business expense detail and then it gets the SDK_BUS_EXP_PER collection.

4. Use the Item method to get SDK_BUS_EXP_PER.

5. Get the SDK_BUS_EXP_DTL collection.

6. Get the SDK_BUS_EXP_DTL, using the InsertItem method. Set the properties.

7. Execute the Save method.

**SDK_BUS_EXP_SaveBusinessExpensePeriod.asp**

This ASP file enables you to update the end date of an existing expense period, and provide options to insert or delete expense details.

To use the SDK_BUS_EXP_SaveBusinessExpensePeriod.asp page:

1. Get the connection information forwarded from the previous page.

2. Get the key fields.

3. Save the business expense period for the selected employee ID and business expense period. Get the specific business expense period by using the Item method. Set the SDK_EXP_PER_DT property with the new value and execute the Save method.

**SDK_BUS_EXP_Signon.asp**

This section describes the behavior of the SDK_BUS_EXP_Signon.asp sample.

This ASP file enables the user to provide signon information and connect to the application server. When you provide and submit the form action, the code in SDK_BUS_EXP_GetSearchParameters.asp is invoked. When you click the Submit button, the JavaScript function checkRequiredFields() runs, which checks whether all the connect information is provided. The connection information is forwarded to the next page, using hidden fields.

**SDK_BUS_EXP_UpdateBusinessExpensePeriod.asp**

This ASP file enables you to update or delete expense period details data from and existing expense period.

To use the SDK_BUS_EXP_UpdateBusinessExpensePeriod.asp page:

1. Get the connection information forwarded from the previous page.

2. Get the key fields.

3. Update business expense details for the selected employee ID and business expense period
4. Get the business expense period to modify using Sub updateBusinessExpensePeriod.

Submitting updateBusinessExpensePeriod gets the business expense period collection. It passes PERIODNUM to the Item method to get the business expense period to be modified.

This page also lists the business expense details, using the getBusinessExpenseDetails function.

5. Use the Save button to save the changes and submit the form.

The Save button uses the JavaScript function saveBusinessExpensePeriod to save changes made to the business expense period. The function saveBusinessExpensePeriod sets the form action to SDK_BUS_EXP_SaveBusinessExpensePeriod.asp and submits the form.
Chapter 9

Using the Excel-to-Component Interface Utility

This chapter provides an overview of the Excel-to-Component Interface utility and discusses how to:

- Run the Excel to Component Interface utility.
- Set up connection information.
- Create templates.
- Enter data on the data input sheet.
- View staged data.
- Create SOAP/XML requests.
- Send requests.
- Receive responses.
- Diagnose and resolve errors.
- Add new languages.

Understanding the Excel-to-Component Interface Utility

Use the Excel to Component Interface utility and component interfaces to upload data from Microsoft Excel into PeopleSoft databases. Each source workbook contains both worksheets and Excel Visual Basic code modules that execute business logic for each transaction.

Use the Microsoft Excel workbooks as a template to create worksheets that are specific to the business logic that you need to use when you are uploading data to the PeopleSoft system. You can copy the data input sheet to other workbooks for distribution without copying the code modules.

The code formats spreadsheet data into a PeopleSoft readable Document Object Model (DOM) structure, and submits it to the PeopleSoft database. Next a PeopleCode program parses the DOM structure and uses the component interface to create entries in the PeopleSoft database, validating the data submitted against the business logic that is built into the PeopleSoft component. Because the component interface is a wrapper around the component, all logic applied during data entry is applied when you are loading data through this tool.
The component interface executes all the necessary PeopleCode events and the field-level edits. Based upon results from saving the component interface, another DOM is created in the PeopleCode that returns success, warnings, errors, or a combination of the three to the Microsoft Excel document. Records in error can be corrected and resubmitted.

**Prerequisites for Using the Excel to CI Utility**

To use the Excel to CI utility you must have the following software installed:

- **Microsoft Excel.**
  The Excel to CI utility is compatible with any of the following versions of Microsoft Excel:
  - Microsoft Visual Basic 6.0 SP5: Run-Time Redistribution Pack.
  You can download this software from the Microsoft website.
- **Microsoft Core XML Services (MSXML) 6.0 or higher.**
  You can download this software from the Microsoft website.

**Microsoft Excel Column and Row Restrictions**

If you are using Microsoft Excel 2003 or earlier version you should be aware of column and row restrictions when using the Excel to CI utility.

**Note.** The information in this section pertains to users running Microsoft Excel 2003 and earlier versions.

Microsoft Excel 2007 does not have any column of row restrictions.

Microsoft Excel 2003 and earlier spreadsheets have a physical limitation of 252 columns and 65,000 rows. When you are creating a template and submitting data, you will need to keep these restrictions in mind.

When a template has more than 252 properties, you can work around this limitation by:

- Deselecting an appropriate number of columns to meet the restriction before submitting to the database. You can determine in advance which fields should be selected as input cells, which cells should be included for submission, and which cells need not be included at all.
- Use wrapped templates.

To work around the row limitations of Microsoft Excel 2003 and earlier versions, you may need to stage data in batches so that the number of rows on the Staging and Submission sheet does not exceed 65,000 rows.

**Note.** You will find the optimal performance when using this utility with small to medium-complexity component interfaces. For large component interfaces, other methods of uploading may be more appropriate, including File Layout and PeopleSoft Application Engine.

In addition, because of the high memory requirements of Microsoft Excel, performance with more complex component interfaces will be inhibited. The number of cells containing values on any one sheet should be less than 1,000,000. Failure to ensure this can potentially result in serious performance issues with Microsoft Excel, including crashes.

---

**Understanding Building Component Interfaces for the Excel to Component Interface Utility**

To use the Excel to Component Interface utility effectively, you must have a complete understanding of the component that you are using and the component interface that is built around it. In addition, you should know what data needs to be entered and which fields on the component need to be exposed as component interface properties. Fields that are not relevant for data input should not be exposed on the component interface. This reduces processing time when you are loading data, as well as saving time when you are building the template because no need to delete unnecessary properties on the template will exist.

Some component interface structures will need to be modified before they can be used to load data through the utility. Components that have logic to insert multiple rows in child collections, and then require more values to be set on those collections, will need modification to the component to work with the Excel to Component Interface utility. Change the component so that the logic to insert and partially populate these rows does not happen by default through the component interface.

%%CompIntfc and %%CompIntfcName can be used so that this logic does not fire either from any component interface or from the component interface that you created for use with the Excel to Component Interface utility.

Additionally, components that have no keys at level 0, but rely on logic at level 0 to load the level 1 collection, cannot be loaded by using the Excel to Component Interface utility.

Component interfaces that rely on CommitWork to save the data cannot be used in the Excel to Component Interface utility.

Prompt and translate table values are validated when data is saved and submitted to the database through the Excel to Component Interface utility. This is different from the behavior on the page when prompts and translates are validated interactively. Some components may use prompts that are dynamically populated. For those situations, you must know what the valid values for the prompt will be.

**Note.** Remember that any changes made to the structure of a component interface will also need to be reflected in the template. Always ensure that the component interface and the template in the Excel to Component Interface utility are in sync. Structural changes made in only the component interface will cause an error in the Excel to Component Interface utility when data is submitted to the database.
Testing Component Interfaces

Before using the Excel to Component Interface utility run the component interface through the component interface tester in three-tier mode. Testing the component interface enables you to troubleshoot any problems before running the component interface through the utility. If the component interface does not work in the tester, it will not work in the Excel to Component Interface utility either. The component interface tester is located on the Tools menu in PeopleSoft Application Designer.


Performance Expectations

The performance of a component interface depends entirely upon the underlying component. If the component has a complex user interface with many pages and scrolls, the component interface generally will have a slower processing time. The best performance times are found with small and medium-complexity component interfaces.

PeopleCode Behavior and Limitations

Certain PeopleCode functions and events that are specific to the user interface do not execute through the component interface. You will need to modify PeopleCode for the component, pages, and records when you build the component interface for the component.

PeopleCode events and functions that relate exclusively to the page interface and online processing cannot be used by component interfaces. These include:

- Search dialog processing.
- Menu PeopleCode and pop-up menus.
- Transfers between components, including modal transfers.
- Dynamic tree controls.
- ActiveX controls.
- DoSave and DoSaveNow.
- Functions that are ignored in a component interface call.


Default Properties

When you create a new component interface in PeopleSoft Application Designer, the system can create default properties for all the fields exposed on the component interface that meet certain criteria.

When you are creating a new component interface, the following requirements must be met to qualify as a default property.
The fields should be of the following types:

- Character
- Long character
- Number
- Signed number
- Date
- Time
- Datetime

The field should be one of the following page control types and must be exposed on the page:

- Edit box
- Drop-down list box
- Check box
- Radio button


---

Running the Excel to Component Interface Utility

This section discusses how to:

- Grant access to the WEBLIB_SOAPTOCI iScript.
- Enable the Developer menu in Microsoft Excel 2007.
- Enable macros in Microsoft Excel.
- Start the Excel to Component Interface utility.
- View the Excel to Component Interface utility coversheet.

See Also

Chapter 9, "Using the Excel-to-Component Interface Utility," Prerequisites for Using the Excel to CI Utility, page 106

---

Granting Access to the WEBLIB_SOAPTOCI iScript

To use the Excel to Component Interface utility, you must grant access to the iScript WEBLIB_SOAPTOCI in the permission list of the user who is building the template.
Enabling the Developer Menu in Microsoft Excel 2007

The Developer menu in Microsoft Excel contains options to work with Microsoft Visual Basic, macros, sheet properties, and so on.

In Microsoft Excel 2007 the Developer menu is not automatically enabled and does not appear on the menu ribbon in the default view of the Excel workspace. In the other versions of Microsoft Excel supported for use with the Excel to Component Interface utility, the Developer menu appears by default.

The following example shows the menu ribbon that appears in the default Microsoft Excel 2007 workspace view:

![The menu ribbon that appears in the default Microsoft Excel 2007 workspace.](image)

To use the Excel to Component Interface utility, you need access to some of the features accessed through via the Developer menu, and therefore you must enable the menu. The following example shows Microsoft Excel 2007 workspace with the Developer menu enabled on the menu ribbon:

![The Developer menu enabled on the Microsoft Excel 2007 menu ribbon.](image)

Once enabled, the Developer menu appears on the far right on the menu ribbon.

To enable the Developer menu in Microsoft Excel 2007:

2. In the upper left corner of the workspace, click the circular Microsoft Office icon. The Recent Documents menu appears.
3. Click the Excel Options button at the bottom of the menu. The Excel Options page appears.
4. In the Top Options For Working with Excel section, check the Show Developer Tab in the Ribbon option.
5. Click the OK button.

The Microsoft Excel 2007 workspace appears and the Developer menu appears on the menu ribbon.
Enabling Macros in Microsoft Excel

The Excel to Component Interface utility relies on macros; therefore, you must enable macros in Microsoft Excel for the utility to work. When a Microsoft Excel spreadsheet is opened, the system displays a dialog box asking you to select whether to enable macros on the spreadsheet. Always select Enable Macros so that the macros delivered with the Excel to Component Interface utility can function.

To ensure that the macros are available to run, you must set the security level in Microsoft Excel to allow macros to open.

To enable macros in Microsoft Excel:
1. Open the Excel to Component Interface utility.
2. From the Excel menu, select Tools, Macros, Security.
3. Select either Medium or Low to enable the macros.
4. Select OK.

Starting the Excel to Component Interface Utility

The Excel to CI utility spreadsheet is located in the PS_HOME/excel directory.

Two versions of the spreadsheet are provided. The version to use depends on the version of Microsoft Excel that you have installed. The following table lists the spreadsheet to use based on the version of Microsoft Excel installed.

<table>
<thead>
<tr>
<th>Microsoft Excel Version Installed</th>
<th>Excel to CI Spreadsheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Microsoft Excel 2000.</td>
<td>ExcelToCI.xls</td>
</tr>
<tr>
<td>• Microsoft Excel 2003.</td>
<td></td>
</tr>
<tr>
<td>Microsoft Excel 2007.</td>
<td>ExcelToCI2007.xls</td>
</tr>
</tbody>
</table>

Viewing the Excel to Component Interface Coversheet

The coversheet of the Excel to Component Interface utility workbook gives a brief overview of the process flow and functionality of the tool.

Access the Coversheet tab in ExcelToCI.xls:
Excel To CI Utility

The purpose of this workbook is to upload data from Excel into PeopleSoft using the Component Interface to execute business logic for each transaction. This source workbook contains both worksheets and Excel VBA code modules. The Worksheets can be copied to other workbooks for distribution without copying the code modules.

Process:

1) **Connection Information sheet**: The information provided on this sheet is required to create a new template or submit data to the database. You will need to specify environment information as well as information in regard to how each transaction should be handled. The Action will be filled in automatically.

2) **Template sheet**: Here you create the template that you are going to use. The template is based upon the structure of a Component Interface on the PeopleSoft database.
   2.1. **New Template**: When prompted, enter the name of your PeopleSoft User ID, password, and the Component Interface you wish to use. The connect information previously provided is used to retrieve the PeopleSoft Component Interface properties. The Component Interface structure is displayed graphically.
   2.2. **Manipulate Template**: Here input cells are selected for inclusion on the Data Input and Staging & Submission sheets. The purpose of each button provided to allow manipulation of the template is more fully described by mousing over the button on the toolbar.
   2.3. **New Data Input**: This button copies the selected input cells into the Data Input sheet. The Data Input sheet becomes active, and you will be prompted before the structures and data on that sheet are deleted.

3) **Data Input sheet**: Here you enter data values for submission to the PeopleSoft database.
   3.1. **Stage Data for Submission**: The data entered on the Data Input sheet is then staged in hierarchical form in preparation for submission to the database.

4) **Staging & Submission sheet**: The last step is to format the data and submit to the database.
   4.1. **Submit Data**: This submits the data to the database. Your PeopleSoft User ID and Password will be required. Each row submitted is marked with the reply from the database which will either be OK, Warning, or Error.
   4.2. **Post Results**: This posts the status for each row submitted to the database on the Data Input sheet so that data in a status of error can be corrected and resubmitted.

Coversheet tab

**Setting Up Connection Information**

This section discusses how to:

- Enter connection information.
- Connect to the database to create a template and submit data.

**Entering Connection Information**

Access the Connect Information tab in ExcelToCI.xls by clicking the Connection Information tab:
Connection Information tab

The information on this page is required to create a new template or to submit data to the database. You will need to specify environment information as well as information about how data should be transmitted. The Action field will be populated automatically based on your setup and the component interface that the template is associated with.

The initial connection settings will be the PeopleSoft default values. You will need to modify these values for your specific implementation of PeopleSoft. If you are unsure what to enter for these values, check with your system administrator.

The connection options are:

**Web Server Machine Name**

The name of the PeopleSoft web server to which you are connecting.

**Protocol**

The protocol used to access the web server. The default is HTTP. The preferred protocol is HTTPS.
<table>
<thead>
<tr>
<th><strong>HTTP Port</strong></th>
<th>The HTTP port number that the web server uses. The default is 80.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Portal</strong></td>
<td>The name of the portal you are using. <em>EMPLOYEE</em> is a default portal shipped with PeopleSoft.</td>
</tr>
<tr>
<td><strong>PeopleSoft Site Name</strong></td>
<td>The PeopleSoft site name that you entered when you installed the PeopleSoft Internet Architecture. The default is <em>ps</em>.</td>
</tr>
<tr>
<td><strong>Node</strong></td>
<td>The PeopleSoft default local node name. The default is <em>PT_LOCAL</em>.</td>
</tr>
<tr>
<td><strong>Language Code</strong></td>
<td>The code for the language in which the data is submitted to the database. If no language code is specified, the base language is used. See Chapter 9, &quot;Using the Excel-to-Component Interface Utility,&quot; Translations and Multilingual Support, page 116.</td>
</tr>
<tr>
<td><strong>Chunking Factor</strong></td>
<td>The number of rows of data to be transmitted to the database at one time. The default is 40.</td>
</tr>
<tr>
<td><strong>Error Threshold</strong></td>
<td>The total number of errors that are permitted before submission to the database ceases. When the error threshold is exceeded, an error message appears and submission to the database stops.</td>
</tr>
</tbody>
</table>
The value for this field is supplied by the system when the component interface is retrieved from the database. However, you can change the supplied value by selecting it from the Action drop-down list.

The types of actions available are based on the structure of the component interface. The actions are:

- **Create.**

  This option is available if the component interface has create keys. Use this mode when new keys are being added at level 0.

- **Update.**

  This option is available if the component interface does not have create keys. Use this mode if you are adding new children to an existing parent.

- **UpdateData.**

  Use this option to update specific non-key values that already exist. The system uses the keys to locate the row, and when a match is found, the row is updated with new data. If a key match is not found by the system, it displays an error message indicating which collection was missing a key match.

  When using the UpdateData action, you must provide all keys for the collection for the system to modify the data.

**Note.** If you want to insert an effective-dated collection at Level 1 containing a child collection at Level 2, you may need to use UPDATE to insert the parent row at level 1 and then use UPDATEDATA to insert the child row at level 2. This is because child rows are copied forward from the current effective-dated collection as a result of the insertion of a new effective-dated parent row. These child rows will be updated by the component processor with the new effective date, and may have the same level 2 keys as the Level 2 child row that you are trying to submit from ExcelToCI.


**Error Thresholds and Chunking**

A running error count is kept for each chunk of data that is being submitted to the database. When the total error count exceeds the error threshold that you specified on the Connection Information tab, submission to the database stops and the system displays an error message. Rows that errored out will have a status of *Error* on the data input page and should be corrected. The data submitted to the database before the error threshold was reached will remain in the target database. Rows not yet submitted will be submitted when the data is restaged and submitted.
Translations and Multilingual Support

You can use the Excel to Component Interface utility to upload data from any installed language. The Excel to CI utility delivers separate Excel macros for each delivered language. The macros contain the translated strings used as labels on the main spreadsheet. The macros are located in the appropriate language directory found in the PS_HOME/EXCEL directory.

Enabling Non-English Languages

To enable a non-English language in the Excel to CI Utility:

- Change the language code on the spreadsheet Connect Information tab to the language to the language you want to use.
- Change the reference to the related language macro to be used, as the default macro contains English language strings.

To change the related language macro, in Excel select the Tools, Macro and right-click on the Visual Basic Editor option. Once in Visual Basic, select Tools, References, and click on the RelLangMacro entry. Change the file to be used to the one with the same name but located inside the translated language directory of your choice. Click OK and then save the change.

- If you are using a language in which a different character set or numeric formatting is used, you need to set the locale of your client machine to match that language. To do so, open Control Panel, Regional settings, and select the correct language and input locale.

Connecting to the Database to Create a Template and Submit Data

Your PeopleSoft login information is needed for both creating the template and submitting data to the database.

Access the Login dialog box by selecting the Template tab and then clicking the New Template button, or by clicking the Submit Data button on the Staging and Submission tab:

The system uses your user ID and password to ensure that you have the correct permissions to access the component interface that you are creating the template on. You must be granted permission to access the component interface that you are using.
**Creating Templates**

The template page is a graphical representation of the component interface structure that you will be using to load data. The structure of the component interface is retrieved from the database when a new template is built. All of the fields that are exposed through the component interface appear on the template page. Fields that are read-only on the component interface will not appear on the template.

The new template macro builds the parent-child relationship within Microsoft Excel based upon the component interface scroll-level definition. The system adds a new row for each scroll level and assigns a unique identifier to it.

Access the Template tab in ExcelToCI.xls to create your template:
Template tab

**Collection**

The name of the component interface collection. A collection is a property that points to a scroll, rather than a field, in the underlying component for a component interface.

**Property**

The component interface property name. Typically, this is also the name of the field on the page.
Record Type

This number represents the parent/child relationship of the records. The level 0 scroll record is always represented by 000. Level 1 scroll records appear with numbers that start with 100 and always have 00 as the last two numbers.

Level 2 scrolls are identified by numbers that start with the identifier of their level 1 parent and end with a 0.

Level 3 scrolls are identified by the first number from the level 1 parent, the second number from its level 2 parent, and then the third number from its own position in the list.

The numbers for each scroll level are incremented based on the number of records that exist at that level. For example, level 0 would be 000, level 1 would be 100, level 2 would be 110, and so on.

**Note.** Component interfaces that have more than 10 collections at a given level will be incremented with alphabetic identifiers. For example, 800, 900, A00, and so on.

Field Type

The standard PeopleSoft type for the field, for example, *Date, Character*, and so on.

Field Length

The length of the field as defined by PeopleSoft. For numeric fields and signed number fields, the length is broken down into integer and decimal positions. For example, a length of 15.3 indicates 15 integer positions and three decimal positions.

Key/Required

If the field is a key field, the system will display a *Y* to the left of the forward slash. When the field is not a key, it will be blank. If the field is a required field, the system will display an *R* to the right of the forward slash. When the field is not required, it will be blank. This information comes from the record definition itself.

**Note.** Fields that are either keys or required *must* be set in order to submit data successfully.

Sequence

The sequence number represents the property order in the template.

Status

This field displays the load status on the Staging and Submission page.

Line Number

This corresponds to the line number on the Input Data and the Staging and Submission pages.

**Understanding the Template Actions Toolbar**

The template actions toolbar is made up of buttons that you use to create and modify a template, as well as create a data input sheet. You can resize the toolbar and move it to any location on the page or even drag it onto the existing standard Microsoft Excel toolbar. When you have moved the toolbar to a location, it will remain there until you move it again. You do not need to move the toolbar each time you open the workbook.

Each button on the toolbar has help text that describes the purpose and use of each of the buttons when you place the cursor over the button.
New Template  Builds a new template based upon a component interface. The New Template macro builds the parent-child relationship within Microsoft Excel based upon the component interface structure.

When you build a new template, the system prompts you for your sign in information.

New Data Input  Builds a new data input sheet based upon the selected input cells. When you build a new data input sheet, the system prompts you as to whether you want to overwrite the existing sheet. If you select Yes, a new data input sheet is created, overwriting the former one.

Select Input Cell  Selects an individual cell to be included in the data input sheet. Cells that have been selected as input cells are highlighted in pink.

Select All Input Cells  Selects all properties to be included in the data input as input cells. When a cell is selected as an input cell, it is highlighted in pink.

Restore Input Cells  Restores the template to its original state and clears default values. The fields in the template will be highlighted in gray, indicating that nothing is included for submission.

Insert New Child  Copies the selected row to be inserted as a new child. This creates multiple occurrences of the same record type.

For example, if the selected row has a template identifier of 100, a new row is inserted that also has an identifier of 100 and is an exact duplicate of the selected row.

Note. Use Insert New Child when multiple children must be submitted under the same parent record. Multiple children should not be created at identifier 000.

Include All for Submission  Includes all properties on the spreadsheet to be included for submission to the database. Cells that are included for submission appear only on the Staging and Submission sheet and do not appear on the data input sheet. Properties that are included for submission are highlighted in blue.

Include for Submission  Includes a single property to be included on the Staging and Submission sheet. Properties that use default values from the template must be included for submission. Cells that are included for submission generally are properties that contain default values or properties that you would like to see in the structure of the Staging and Submission sheet. Properties that are included for submission are highlighted in blue.

Deselect Input Cell  Changes a cell that was previously selected as an input cell to a cell that is included for submission. The cell is no longer included on the data input sheet but appears as part of the structure on the Staging and Submission sheet.

Clear Template  Clears all the data and structures on this sheet.
Do Not Include for Submission

Does not include the selected property for submission to the database. If a property is not included for submission, it will not appear in the structure that is submitted to the database on the Staging and Submission sheet. Properties that are not included for submission will appear only on the template worksheet and are not submitted to the database. Properties that are not included for submission are highlighted in gray.

---

Note. When you create a new template or a new data input sheet, the system clears the existing worksheet of all existing information. If you have a template or data input sheet that you need to save from previous uploads, save a copy of the worksheet before you create a new template or data input sheet.

---

**Entering Data into the Template**

When determining which properties to include as input cells and which properties to include for submission, remember that the component interface uses the same business logic and executes the same PeopleCode as if the record were entered online using the page in your PeopleSoft application. To provide the minimal data necessary, these fields must be provided either with default (hard-coded) values or values that you provide using the data input sheet.

---

**Adding a New Child Record**

By default, each collection is represented once on the template. To insert copies of a given collection, select that collection and click the Insert New Child button to create a copy of the selected row. The collection that you selected is copied so that you can have two rows under the same parent.
Adding Default Values

Some fields have default values associated with them, either in the record definition or at runtime when the record is created on the database. Additionally, many components trigger PeopleCode, which supplies default values, as well. To accept the database default, include the property for submission and the system default will be used.

Some fields may exist for which you want to create your own default. For example, if you want to set the value of a field named Status as of Effective Date to A for every row that you submit, enter that value for the field in the template. Then include the cell for submission on the template. The field will not appear on the data input page, but the value will appear in the field on the Submit to Database page. This is useful for effective dates, status fields, set IDs for simple imports, and so on.

When providing values for translate fields or prompt tables, provide the field value rather than the short or long description for the translate value. If you are unsure of the field values, check in the record or field definition in PeopleSoft Application Designer.

Template Wrapping

The information in this section is applicable if you are using Microsoft Excel 2003 or earlier versions.

Template Wrapping and Large Component Interfaces

Excel to Component Interface templates for Microsoft Excel 2003 and earlier versions can load component interface definitions containing up to 502 properties, by splitting the metadata for the component interface properties that exceed the 254-column limit. The template will appear split in two sections, or wrapped.

Inserting Child Rows in Wrapped Templates

When using the Insert New Child toolbar action on template rows which have child rows, the new parent-child rows should maintain the same hierarchical order as the original parent-child rows.
Exercise caution when you insert child rows in a wrapped template. These templates have hidden rows which are clones of the visible rows. If you insert a child row on the visible row of the wrapped template, verify that it also was duplicated in the hidden section above the wrapped portion. If it was not, copy it manually there as well in order to preserve the structure.

**Entering Data on the Data Input Sheet**

The data input sheet enables you to enter data into the Excel to Component Interface utility so that it can be loaded to the database by using the component interface that you've selected. You can enter data manually or you can cut and paste it from another spreadsheet or third-party application.

**Using the Data Input Sheet**

Access the data input tab to enter data:
Data Input tab

The field labels that appear on the data input sheet are those properties that you selected as input cells on the template. Each scroll level is identified by color. The record type from the template is also displayed for each property.

The system creates default date, datetime, and number formats when it creates the template. You can modify this format by using default cell formatting of Microsoft Excel when entering data, with the exception of the d/m/yy format for dates and datetimes. Instead, always use a d/m/yyyy format for dates and datetimes. To access the formatting feature, select Format, Cells from the Excel menu.

The data input sheet is also used to correct data that failed to submit to the database. Errors that are flagged on the Submit to Database page are posted to the data input page, and when you have corrected them, the items marked in error can be staged again to the Staging and Submission sheet.

**Data Input Actions**

The data input Actions toolbar contains the Stage Data for Submission button, which takes the data that you entered on the data input sheet and stages it for submission to the database. When the data is staged, it appears on the Staging and Submission sheet in the hierarchical template structure. At this point, you should check that all fields are populated as expected. When the data is staged, it displays both the data on the data input sheet and the data that you specified as default values.
Viewing the Staged Data

Access the Staging and Submission tab:

Note. Microsoft Excel has a physical limitation of 252 columns and 65,000 rows. Depending on the complexity of the record structure and the number of scroll-level records that are included in the template, you can use multiple rows on the Staging and Submission sheet to reflect one row of data from the data input sheet. To work around the limitations of Microsoft Excel limitations, you may need to stage your data in batches so that the number of rows on the Staging and Submission sheet will not exceed 65K.

Staging and Submission Actions Toolbar

**Post Results**

The results of the submission are copied to the data input sheet, where you can view the status of each row that is submitted and make any necessary corrections to rows that have the status of **Error**.
Submit Data
The login dialog box appears. You must specify your user ID and password.
The system submits the data to the database in the chunks that you specified on
the Connection Information sheet.
After correcting any errors on the data input sheet, you can submit the data again.
The items that had been marked as Error will be resubmitted, whereas those
marked OK and Warning will be ignored.

Error When Submitting Existing Keys
If you receive the error message Row already exists with the specified keys and you are in CREATE mode,
the key already exists at level 0 or is part of the search record.

To verify that the key exists:
1. Open the component interface in PeopleSoft Application Designer.
2. Launch the component interface tester by selecting Tools, Test Component Interface.
   The component interface Tester search dialog box appears. This dialog box displays the keys (in the left-
   hand columns) for getting, creating, or finding an instance of the component interface.
3. Enter the value for the key that you are testing.
4. Click Get Existing for the key that you are about to add, using the Excel to component interface utility.
   If the Get Keys command returns the key, the key already exists and you must add data by using
   UPDATE mode.
   If you receive a message that no row exists for the key, then the key does not exist at level 0 and the data
   should be added by using CREATE mode.


Correcting and Resubmitting Data
After you submit the data to the database, results of the process appear on the Staging and Submission sheet.
If a submission had an error, the errored status appears on the Staging and Submission sheet. Use the data
input page to correct the data and then resubmit it to the database. Continue this process of correcting errors
and resubmitting the data until no errors remain.

Note. Data that was not submitted because the error threshold was reached will have no status. When the data
that created the error is corrected on the data input sheet, the data that was not submitted will be staged to the
database.

Submission Statuses
Errors received for each record submitted appear in a comment field when you move the cursor over the
status column. The records marked OK in green have been successfully submitted and cannot be restaged for
submission and can be kept as a record of work completed.

One of the following three status values can appear when you submit data to the database:
Ok

The submission to the database finished successfully. The field is highlighted in green.

These records cannot be restaged for submission.

Warning

The data was saved to the database successfully, but a warning was generated in the process. The field is highlighted in yellow.

Error

The data was not saved to the database due to an error. This field is highlighted in red.

To see the error message that the component interface generated, place your cursor over the Status field. This message indicates how the data needs to be corrected.

Creating SOAP/XML Requests

You can construct a SOAP/XML (Simple Object Access Protocol/Extensible Markup Language) request to create, update, or get component interface rows. The request and response contain component interface data in a SOAP/XML format.

Request Format

The following example shows the request format:

```xml
<?xml version="1.0"?>
  <SOAP-ENV:Body>
    <Action__CompIntfc__CIName>
      Tags and Data
    </Action__CompIntfc__CIName>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Valid actions are Create, Get, Update, and Updatedata.

CIname is the name of the component interface.

Tags and Data contains the tags and data for the component interface row or rows.

Sample Create Request

The following example shows a Create request:
Sample Get Request

The following example shows a Get request:

<?xml version="1.0"?>
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <Get__CompIntfc__SDK_BUS_EXP>
   <SDK_EMPLID>8052</SDK_EMPLID>
  </Get__CompIntfc__SDK_BUS_EXP>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Sample Update Request

The following example shows an Update request:

<?xml version="1.0"?>
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <UPDATE__CompIntfc__SDK_BUS_EXP>
   <SDK_EMPLID>8001</SDK_EMPLID>
   <SDK_BUS_EXP_PER>
    <SDK_EXP_PER_DT>08/14/2002</SDK_EXP_PER_DT>
    <SDK_BUS_EXP_DTL>
     <SDK_EXP_DT>08/14/2002</SDK_EXP_DT>
     <SDK_EXPENSE_CD>01</SDK_EXPENSE_CD>
     <SDK_EXPENSE_AMT>1234.56</SDK_EXPENSE_AMT>
     <SDK_CURRENCY_CD>USD</SDK_CURRENCY_CD>
     <SDK_BUS_PURPOSE>Client Visit</SDK_BUS_PURPOSE>
     <SDK_DEPTID>10100</SDK_DEPTID>
    </SDK_BUS_EXP_DTL>
   </SDK_BUS_EXP_PER>
  </UPDATE__CompIntfc__SDK_BUS_EXP>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Sample Updatedata Request

The following example shows an Updatedata request:
Sending Requests

To send the request, post the XML code to the URL of the PeopleSoft Pure Internet Architecture server with the appropriate path to the iScript on the server.

**Note.** The PeopleSoft user ID and password must be sent in the SOAP request header with the identifiers of `userid` and `pwd`. You should send the request on a secure site.

Use this format:

```
Protocol(http or https)>://<WebServerMachineName>:<HTTPPort>/psc/ps/<Portal>⇒
/Node>/s/
WEBLIB_SOAPTOCI.SOAPTOCI.FieldFormula.IScript_SOAPToCI?&disconnect=y&postDataBin=y
```

- **WebServerMachineName** Machine name of the server.
- **HTTPPort** Port of the server.
- **Portal** Portal defined on the PeopleSoft Pure Internet Architecture server.
- **Node** Node defined on the PeopleSoft Pure Internet Architecture server.

For example,

```
```

Receiving Responses

This section provides examples of response types.
Viewing a Response if a Row Already Exists

This is one example of the error response. The messages vary depending on the type of error.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
  <SOAP-ENV:Body>
    <USER_PROFILE>
      <Error-Warning>
        <Message>
          <Type>Error</Type>
          <MessageSetNumber>91</MessageSetNumber>
          <MessageNumber>49</MessageNumber>
          <MessageText>Row already exists with the specified keys. (USER_PROFILE) (91,49)</MessageText>
          <ExplainText>A row already exists in the database with the specified keys.</ExplainText>
        </Message>
      </Error-Warning>
      <Key_information>
        <UserID>PTDMO10</UserID>
      </Key_information>
    </USER_PROFILE>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Viewing a Sample Get Request and Response

The following XML code gets an SDK_BUS_EXP component interface row for an employee with an employee ID of 8052:

```xml
<?xml version="1.0"?>
<SDK_BUS_EXP action="GET">
  <SDK_EMPLID key="Y">8052</SDK_EMPLID>
</SDK_BUS_EXP>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The XML response for this employee is:
<?xml version="1.0" encoding="UTF-8" ?>
  <SOAP-ENV:Body>
    <SDK_BUS_EXP>
      <SDK_BUS_EXP_PER>
        <SDK_EMPLID>8052</SDK_EMPLID>
        <SDK_EXP_PER_DT>2000-11-09</SDK_EXP_PER_DT>
      </SDK_BUS_EXP_PER>
      <SDK_BUS_EXP_DTL>
        <SDK_EMPLID>8052</SDK_EMPLID>
        <SDK_EXP_PER_DT>2000-11-09</SDK_EXP_PER_DT>
        <SDK_CHARGE_DT />
        <SDK_EXPENSE_CD />
        <SDK_EXPENSE_AMT>0</SDK_EXPENSE_AMT>
        <SDK_CURRENCY_CD>USD</SDK_CURRENCY_CD>
        <SDK_BUS_PURPOSE />
        <SDK_DEPTID />
      </SDK_BUS_EXP_DTL>
    </SDK_BUS_EXP>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Diagnosing and Resolving Errors

This section discusses how to:

- View the log files.
- Resolve error messages for client environments.

Viewing Log Files

If you select the check box to create log files when building a template or submitting to the database, two log files are created—one that logs the activity of ExcelToCI.xls and the other that logs the SOAPTOCI Web Library.

The log for ExcelToCI.xls is created in the temp directory on the workstation running the Microsoft Excel spreadsheet.

The log for the Web Library, SOAPTOCI<unique_number>.log, is created on the application server in the <PS_CFG_HOME> directory. This file contains both the SOAP request and the SOAP response.

Log files are written for each chunk of data that is sent to the database.

Resolving Error Messages for Client Environments

The following table lists common errors and error messages and their possible resolutions.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Possible Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component not correctly registered</td>
<td>Reinstall Visual Basic 6.0 SP5: Run-Time Redistribution Pack found on the Microsoft download site.</td>
</tr>
<tr>
<td><strong>Error Message</strong></td>
<td><strong>Possible Resolution</strong></td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ActiveX component not correctly registered (Error 336)</td>
<td>Reinstall Visual Basic 6.0 SP5: Run-Time Redistribution Pack found at the Microsoft downloads website.</td>
</tr>
<tr>
<td>Error Number: -2147024770 Description: Automation error. The specified module could not be found.</td>
<td>Perform the following steps:&lt;br&gt;1. Open Windows Explorer.&lt;br&gt;2. Navigate to c:\winnt\system32 directory and locate msxml4.dll.&lt;br&gt;3. Right-click the DLL and select Register COM Server. The message <code>DLLRegisterServer in c:\winnt\system32\msxml4.dll succeeded.</code> will appear. 4. Click OK.</td>
</tr>
<tr>
<td>Error Number: 429 Description: ActiveX component can't create object.</td>
<td>Perform the following steps:&lt;br&gt;1. Open Windows Explorer.&lt;br&gt;2. Navigate to c:\winnt\system32 directory and locate msxml4.dll.&lt;br&gt;3. Right-click the DLL and select Register COM Server. The message <code>DLLRegisterServer in c:\winnt\system32\msxml4.dll succeeded.</code> will appear. 4. Click OK.</td>
</tr>
<tr>
<td>Error Number: -214722099 Description: Automation error in the dll.</td>
<td>Perform the following steps:&lt;br&gt;1. Location and open the file ExcelToCI.xls.&lt;br&gt;2. Press Alt + F11 to open the Microsoft Visual Basic Editor.&lt;br&gt;3. Select Tools, Add references.&lt;br&gt;4. Deselect anything that begins with Microsoft XML.&lt;br&gt;5. Browse for c:\winnt\system32\msxml4.dll and click OK.&lt;br&gt;6. Select that version of msxml and click OK.&lt;br&gt;7. Click Save.</td>
</tr>
<tr>
<td>Not Authorized (90,6)</td>
<td>The user who is trying to access the component interface from ExcelToCI does not have access to the component interface. Please provide access using PeopleTools Security.</td>
</tr>
</tbody>
</table>
Appendix A

Creating Component Interface-Based Services

This appendix provides an overview of creating component interface-based services,

Understanding Generating Component Interfaced-Based Services

PeopleSoft Integration Broker enables you to take an existing component interface and create a service that can invoke the component interface.

Further, it creates service operations, including request messages and response messages (if appropriate). The system creates an inbound any-to-local routing for the service operation version, as well as handlers for each method you choose to include in the service.

All service operations you generate from component interfaces are synchronous service operations.

After you create the service operation you can access the service definition to view and capture the WSDL.

See Enterprise PeopleTools 8.50 PeopleBook: PeopleSoft Integration Broker, "Creating Component Interface-Based Services."
Index

Numerics/Symbols

%Complntfc 107
%ComplntfcName 107
%Session 63

A

actions
create 115
data input 124
staging and submission 125
template actions 119
update 115
updateData 115
ActiveX controls
errors 132
unsupported events 59
alternate search keys 7
Altkey property 26
API See application programming interface
Application Designer
APIs 71
building APIs 65
C++ templates 73
COM 79
Component Interface Tester 49
creating definitions 5
Java templates 66
PeopleCode template 61
using views 5
validating component interfaces 46
application programming interfaces 1
accessing C++ APIs 72
bindings 65
building in Java 65
C++ 71
COM 79
component interface API 3, 51, 52
dummy rows 50
naming rules 31
architecture 3
attributes
collections 5
keys 4
methods 5
name 4
overview 3
properties 4
automatic field defaults 11
autoregister 79

B

backpointers 33
batch processes 19
bindings
C++ 71, 72
COM 80
eyearly-binding 81, 94
Java 65, 66
late-binding 81
PeopleSoft API 65, 80
buffers 14
building a component interface 107
C++
building APIs 71
client setup 72
configuring compilers 72
memory conflicts 58
requirements 72
runtime code templates 73
third-party applications 72
Cancel method 5, 36
cell formatting 124
collection methods 55
collection name 118
collection properties 29
collections
CreateKeyInfoCollection 17
defined 5
empty 14
FindKeyInfoCollection 17
GetKeyInfoCollection 17
overview 33
restrictions 10
search key 16
user-defined 4
collections icon 7
COM (Component Object Model) 39
APIs 79
ASP sample 96
Excel sample 94
memory conflicts 58
overview 1
requirements 80
runtime code templates 81
SDK Excel sample 93
third-party applications 81
Index

- type libraries 80
- CommitWork 107
- compiler configuration 72
- CompIntfcName 20
- component buffer 14
- component interface API 3, 51, 52
- component interface methods
  - standard methods 43
  - user defined 45
- component interfaces
  - adding to a menu 12
  - creating 11
  - exposing fields 14
  - setting security 47
  - testing 48, 108
  - validating 46
- Component Interface Tester
  - editing history items 50
  - Enter key values dialog box 49
  - Find Results dialog box 51
  - getting dummy rows 50
  - history items 50
  - ItemByKeys: Enter parameters dialog box 55
  - ItemByKeys parameters 54
  - modes 50
  - procedures 48
  - Test dialog box 52
  - testing collection methods 53
  - testing methods 52
  - testing properties 52
- component interface view 5, 6
- ComponentName 20
- Component Object Model
  - See COM (Component Object Model)
- component transfers 59
- component view 5
- COM Type Library 79
- Connection Information tab in Excel to Component Interface 113
- connection settings
  - defaults 113
  - Excel to Component Interface 112
  - HTTP port 114
  - login screen 116
  - portal name 114
  - protocol 113
  - web server 113
- Connect method 35
- CopyRowset 61
- Count method 40
- Count property 29
- Coversheet tab in Excel to Component Interface 111
- create action 115
- CreateKeyInfoCollection 17
- create keys 4, 12, 14, 15, 115
- Create method 36
- create new 50
- Create Reference dialog box 32
- CreateSession 34
- CurrentItem collection method 54
- CurrentItem method 42
- CurrentItemNum method 42

D

- data input actions toolbar 124
- data input sheet
  - copying to other worksheets 105
  - creating 120
  - entering data 123
  - limitations 125
  - overview 124
- Data Item property 29
- data submission 125
- date fields 10
- dates 124
- datetime fields 10
- debugging 19, 117
- DecimalPosition property 25
- decimals 119
- defaults
  - automatic 11
  - connection settings 113
  - creating 108
  - Excel to Component Interface 120, 121, 122
  - fields 122
  - properties 10, 12
  - definition name 12
- definitions
  - design-time vs. runtime 5
  - in Application Designer 5
- DeleteItem(index) collection method 54
- DeleteItem(Index) method 41
- deleting
  - child properties 33
  - DeleteItem(index) 54
  - DeleteItem(Index) 41
  - keys 14
  - properties 14, 30
- deleting elements 7
- derived properties 31
- Description property 20
- Deselect Input Cell button 120
- design-time properties 5
- DisableMenuItem 59
- Document Object Model (DOM) 105
- DoModalPageGroup 59
- DoNotIncludeForSubmission button 120
- DoSave() 60
- DoSaveNow() 60
- drop-down list boxes 10
- dummy rows 50
- dynamic tree controls 59

E

- edit boxes 10
- EditHistoryItems 18
- edit history items 50
- Edit Property dialog box 30
- email 57
- EnableMenuItem 59
- enabling macros 94
- entering data 123
- errors
  - correcting 126
  - Data Input tab 124
  - diagnosing 131
  - in Excel to component interface 127
  - list of messages 131
  - log files 131
message log 52
message logs 117, 131
PSProperties not loaded from file 66
setting thresholds 114
submitting existing keys 126
thresholds 115
validation 58
Excel See Microsoft Excel
Excel to component interface
correcting data 126
error messages 131
Excel to Component Interface
building a component interface 107
Connection Information tab 113
data input sheet 124
enabling macros 111
entering connection settings 112
Excel limitations 125
Excel restrictions 107
logging in 116
Login dialog box 116
PeopleCode limitations 108
performance 108
Staging and Submission tab 125
templates 117
Template tab 118

F

field defaults
automatic 11
criteria for 10
FieldName property 22
fields
automatic default criteria 10
decimals 119
defaults 11
exposing 14
integers 119
length 119
names 7
standard types 119
FindKeyInfoCollection 17
find keys 4, 15
Find method 5, 36
Format property 24
functions
Build API names 72
createSession 34
ignored 60
PeopleCode 45
PeopleCode limitations 59
restrictions 108
session 34

g

getComplIntfc 35
GetDummyRow 50
GetDummyRows 21
GetEffectiveItem(DateString, SeqNum) collection method 54
GetEffectiveItem method 42
GetEffectiveItemNum(DateString, SeqNum)
collection method 54
GetEffectiveItemNum method 43
get existing 50
GetHistoryItems 18
get history items 50
GetKeyInfoCollection 17
get keys 4, 15
Get method 5, 37
getOAType() 27
GetPropertyByName method 37
GetPropertyInfoByName(). Enter parameters:
dialog box 53
GetPropertyInfoByName method 38
getType() 27

H

hidden edit validation error 58
HideMenuItem 59
hierarchy 122
HTTPPort 129
HTTP port 114

I

icons 6
Include All for Submission button 120
Include for Submission button 120
infinite processing 57
InsertItem(index) collection method 53
InsertItem method 40
Insert New Child button 120
integration
accessing external systems 3
SDK example 87
Integration SDK
COM ASP sample 96
COM sample 93
install location 87
requirements 88
samples 87
test page 88
using the Java sample 89
InteractiveMode 19
interactive mode
considerations 58
debugging 19
testing component interfaces 50
UNIX servers 19
invalid component interfaces 46
invisible fields 10
IsCollection property 23
iScript 109
IsModalPageGroup 59
IsReadOnly property 26
Item(index) collection method 53
Item(Index) method 41
ItemByKeys(key1, key2, …) collection method 54
ItemByKeys(keys) method 41
ItemByKeys parameters 54
ItemByKeyName method 40
ItemNum property 29
ItemSelected 59
J

Java
  bindings 65
  building APIs 65
  class file 67
  methods 27
  object adapter 27
  requirements 66
  runtime code template 66
Java Development Kit (JDK) 66, 72, 80
Java Virtual Machine (JVM) 66, 72, 80
Jolt failover 68

K

Key property 24
keys
  adding 16
  adding manually 15
  creating 12
  defined 4, 14
  deleting 16
Excel to Component Interfaces 119
  existing 126
  exposing 14
  getting existing records 50
  icon 6
  in scrolls 34
  modifying 15
  removing 16

L

LabelLong property 22
LabelShort property 22
language code 114, 116
languages
  CopyRowset considerations 61
  installed languages for Excel to Component Interface 116
  language codes 114
  submitting in several languages 116
Length property 25
level-zero records 14
limitations
  PeopleCode 108
ListBoxItem property 26
load balancing 68
log files
  generating 117
  viewing 131
logging in 116
Login dialog box in Excel to Component Interface 116
long character fields 10

M

macros

COM sample 94
  enabling 111
  mapping related keys 32
Market property 20
memory
  conflicts 58
  releasing 70
  requirements 107
menu PeopleCode 59
menus 12
message logs 52, 117
messages
  Excel to Component Interfaces 116
  PSMessages collection 58
  session error messages 63
methods
  collection 39
  defined 5
  Java 27
  session 34
  standard 5, 35
  testing 52
  user-defined 5
Microsoft
  Visual Basic 80, 81
Microsoft Excel
  COM sample file location 94
  Excel to Component Interface utility overview 105
  physical limitations 106
  using the COM sample 94
Mobile Agent
  synchronization 55
mobile properties
  derived 31
  do not send updates 31
  send updates 31
modal transfers 59
modifying keys 15
multilingual support 116
multiple instances 58

N

name column 7
Name property 21
naming guidelines 4, 12
New data input button 120
New Template button 120
node 114
Node 129
noninteractive mode 50
number fields 10

O

OaType property 23
object adapter 27
out-of-sync icon 7

P
page control types 10
parent scrolls 34
passwords
  Excel to Component Interface 117
  SOAP request header 129
PeopleCode
  client-only limitations 61
  CopyRowset 61
  creating user-defined methods 45
  generating templates 61
  ignored functions 60
  limitations 59
  templates 63
  trigger order 4
  valid variables 63
PeopleCode limitations 107
peoplesoft_peoplesoft__i.h 71
PeopleSoft Application Designer
  See Application Designer
PeopleSoft site name 114
performance 108
permission lists 47
pop-up menus 59
Portal 129
portal name 114
Post Results button 125
PrePopup 59
Prompt property 25
properties
  automatic defaulting 12
  criteria for automatic defaulting 10
  defined 4, 118
  deleting empty 14
  exposing 14
  limits 106
  ordering 7
  read-only 33
  reference 31
  renaming 31
  standard 4, 16
  testing 52
  user-defined 4, 29
Properties dialog box
  Synchronization tab 56
properties that are keys 6
property count 29
property icon 7
PropertyInfoCollection 21
protocol 113
PSMessages 58

R
radio buttons 10
read-only properties 7, 33
RecordName property 22
record names 7
record type
  defined 119
  reference backpointers 33
  reference paths 33
  reference properties 31
  related display 57
  related keys 32
  requests
  sending 129
required fields 119
Required property 24
required property icon 7
requirements 88
response types 129
Restore Input Cells button 120
row inserts 57
rules 53
runtime code templates 73
runtime exceptions 4
run-time properties 5

S
Save method 5, 37
scope conflicts 57
scroll areas 5, 33
scroll levels 107
SDK 88
SDK_BUS_EXP 87, 89
SDK_BUS_EXPENSES 88
SDK_BUS_EXPENSES page 88
SDK_CL_SAMPLES 88
SDK (Software Development Kit)
  See Integration SDK
SDK Java sample 89
SDK testing 89
search dialog processing 59
search keys 15
security 47
  accessing the SDK 88
  user profiles 1
Select All Input Cells button 120
Select Input Cell button 120
sequence number 119
session functions 34
session methods 34
SetPropertyByName method 38
signed number fields 10
site name 114
SOAP
  log files 131
SOAP/XML requests 127
SOAPTOCI web library
    generating log files 117
Stage for Submission button 124
Staging and Submission Actions toolbar 125
Staging and Submission tab 125
standard methods 5, 35, 44
  disabling 43
  enabling 43
standard methods icon 7
standard properties 4, 16
status 119
StopOnFirstError 19
submission statuses 126
Submit Data button 126
submitting data 125, 126
synchronization 55

template actions toolbar 119
templates
adding child records 121
adding defaults 122
C++ 73
column limitations 122
COM 81
creating in Excel to Component Interfaces 117
entering data 121
Java 67
PeopleCode 61, 63
testing 121
Visual Basic template 82
wrapping 122
Template tab in Excel to Component Interface 118
testing 48
SDK testing 89
third-party applications 72, 81
thresholds for errors 114
time fields 10
TransferPage 59
tree controls 59
trigger order 4
troubleshooting
  hidden edit validation error 58
  infinite processing loops 57
Type property 23

U
UncheckMenuItem 59
UNIX servers 58
update action 115
updateData action 115
user-defined methods 5, 7
  creating 44
  restrictions 12
user-defined properties 4
  creating 29
  deleting 30
  renaming 30
user ID 117
user profiles 1

V
validating 46
variables 63
view columns 7
view icons 6
viewing submission results 125
views 5
Visual Basic 81
  component interface code 82

W
walkpaths 33
warnings 127
WEBL1B_SOAPTOCI 109
WebServerMachineName 129
web server name 113
WinMessage 57