



PeopleTools 8.12 Component Interface PeopleBook

PeopleTools 8.12 Component Interface PeopleBook

SKU MTCIr8SP1 1200

PeopleBooks Contributors: Teams from PeopleSoft Product Documentation and Development.

Copyright © 2001 by PeopleSoft, Inc. All rights reserved.

Printed in the United States of America.

All material contained in this documentation is proprietary and confidential to PeopleSoft, Inc. and is protected by copyright laws. No part of this documentation may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, including, but not limited to, electronic, graphic, mechanical, photocopying, recording, or otherwise without the prior written permission of PeopleSoft, Inc.

This documentation is subject to change without notice, and PeopleSoft, Inc. does not warrant that the material contained in this documentation is free of errors. Any errors found in this document should be reported to PeopleSoft, Inc. in writing.

The copyrighted software that accompanies this documentation is licensed for use only in strict accordance with the applicable license agreement which should be read carefully as it governs the terms of use of the software and this documentation, including the disclosure thereof.

PeopleSoft, the PeopleSoft logo, PeopleTools, PS/nVision, PeopleCode, PeopleBooks, Vantive, and Vantive Enterprise are registered trademarks, and *PeopleTalk* and "People power the internet." are trademarks of PeopleSoft, Inc. All other company and product names may be trademarks of their respective owners.

Contents

About This PeopleBook

Before You Begin	vii
Related Documentation	vii
Documentation on the Internet	viii
Documentation on CD-ROM	viii
Hardcopy Documentation	viii
Typographical Conventions and Visual Cues	viii
Comments and Suggestions	x

Chapter 1

Component Interface

Introduction	1-1
Component Interface Architecture	1-2
Attributes of a Component Interface	1-3
Component Interface Name	1-3
Keys	1-3
Properties and Collections	1-4
Security for Properties	1-5
Methods	1-6

Chapter 2

Creating a Component Interface

Views in Application Designer	2-1
Component Interface View	2-2
Component Interface View Display	2-2
Creating a New Component Interface	2-3
Creating Properties	2-5
Making Properties Read-Only	2-6
Creating Collections	2-6
Adding and Removing Keys	2-7
Which Properties to Expose?	2-7

Guidelines for Exposing Components.....	2-7
Working with Methods	2-8
Standard Methods	2-8
User-Defined Methods	2-9
Setting Component Interface Security	2-10
Testing a Component Interface.....	2-13
Getting the Signature of the ItemByKey Method.....	2-18
Validating a Component Interface.....	2-20
Generating Visual Basic Template	2-20
Generating PeopleCode	2-24

Chapter 3

The Component Interface API

Binding Considerations	3-3
COM Binding.....	3-3
Third Party Application.....	3-3
External API Installation.....	3-4
C Header Binding.....	3-4
Third Party Application.....	3-4
C Header File	3-4
Connecting to a Component Interface	3-5
Installing External Client Settings for the API.....	3-5
Comparing Component Interface and Components.....	3-5
Differences in Search Dialog Processing.....	3-5
Differences in PeopleCode Event and Function Behavior.....	3-5
Limitations of Client-Only PeopleCode.....	3-6
Email from a Component Interface.....	3-6
WinMessage Unavailable.....	3-6
Calling another Component Interface	3-6

Chapter 4

Component Interface Example

PeopleCode Example.....	4-2
Java and Active Server Page Examples	4-4
Active Server Page Example.....	4-6
Connecting to the Application Server	4-7
Getting an Instance.....	4-7

Finding an Existing Record.....	4-8
Getting an Instance of Data.....	4-9
Migrating Through Scrolls.....	4-10
Editing and Accessing Data in an Item	4-11
Inserting a Row into a Collection.....	4-11
Deleting a Row from a Collection	4-12
Disconnecting from a Session.....	4-12
Java Example	4-12
Connecting to the Application Server	4-13
Getting an Instance of the Component Interface.....	4-13
Finding an Existing Record.....	4-13
Getting an Instance of Data.....	4-13
Migrating Through Scrolls.....	4-14
Editing and Accessing Data in an Item	4-14
Inserting an Item into a Collection.....	4-15
Deleting a Row from a Collection	4-15
Disconnecting from a Session.....	4-15

Chapter 5

Component Interface SDK

Requirements	5-1
The PTSDK Development Project.....	5-1
PTSDK Project Objects.....	5-2
PTSDK Records	5-2
SDK_BUS_EXPENSES Test Page.....	5-7
Installing the PTSDK Project.....	5-7
Component Interface Tester and Samples	5-10
C++ Tester and Sample.....	5-10
Preparing Your C++ Tester and Sample	5-11
Using the C++ CI Tester	5-13
Using the C++ CI Sample	5-14
Visual Basic Tester and Sample.....	5-15
Preparing Your Visual Basic Tester and Sample	5-15
Using the Visual Basic CI Tester	5-15
Using the Visual Basic CI Sample	5-16
ASP Tester and Sample.....	5-18
Preparing Your ASP Tester and Sample	5-19
Using the ASP CI Tester	5-19

Using the ASP CI Sample	5-20
Java Tester and Sample	5-22
Preparing Your Java Tester and Sample	5-22
Using the Java CI Tester	5-23
Using the Java CI Sample	5-23

Index

ABOUT THIS PEOPLEBOOK

This book describes a PeopleSoft component interface that is a PeopleTools object that you create in Application Designer. It allows access to a PeopleSoft component for synchronous access from another application. This book includes the following:

Introduction to Component Interface introduces the component interface architecture, including component interface properties, collections, keys, and methods.

Creating a Component Interface describes how to create a component interface.

The Component Interface API discusses techniques for accessing components from PeopleCode and through Visual Basic or web-based applications through the Component Interface API.

Component Interface Example shows a sample of creating and calling a component interface.

Component Interface SDK describes how to use the component interface resources in the Software Development Kit to integrate your PeopleSoft application with third party products.

Before You Begin

To benefit fully from the information covered in this book, you need to have a basic understanding of how to use PeopleSoft applications. We recommend that you complete at least one PeopleSoft introductory training course.

You should be familiar with navigating around the system and adding, updating, and deleting information using PeopleSoft windows, menus, and pages. You should also be comfortable using the World Wide Web and the Microsoft® Windows or Windows NT graphical user interface.

Related Documentation

To add to your knowledge of PeopleSoft applications and tools, you may want to refer to the documentation of the specific PeopleSoft applications your company uses. You can access additional documentation for this release from PeopleSoft Customer Connection (www.peoplesoft.com). We post updates and other items on Customer Connection, as well. In addition, documentation for this release is available on CD-ROM and in hard copy.



Important! Before upgrading, it is *imperative* that you check PeopleSoft Customer Connection for updates to the upgrade instructions. We continually post updates as we refine the upgrade process.

Documentation on the Internet

You can order printed, bound versions of the complete PeopleSoft documentation delivered on your PeopleBooks CD-ROM. You can order additional copies of the PeopleBooks CDs through the Documentation section of the PeopleSoft Customer Connection Web site:
<http://www.peoplesoft.com/>

You'll also find updates to the documentation for this and previous releases on Customer Connection. Through the Documentation section of Customer Connection, you can download files to add to your PeopleBook library. You'll find a variety of useful and timely materials, including updates to the full PeopleSoft documentation delivered on your PeopleBooks CD.

Documentation on CD-ROM

Complete documentation for this PeopleTools release is provided in HTML format on the PeopleTools PeopleBooks CD-ROM. The documentation for the PeopleSoft applications you have purchased appears on a separate PeopleBooks CD for the product line.

Hardcopy Documentation

To order printed, bound volumes of the complete PeopleSoft documentation delivered on your PeopleBooks CD-ROM, visit the PeopleSoft Press Web site from the Documentation section of PeopleSoft Customer Connection. The PeopleSoft Press Web site is a joint venture between PeopleSoft and Consolidated Publications Incorporated (CPI), our book print vendor.

We make printed documentation for each major release available shortly after the software is first shipped. Customers and partners can order printed PeopleSoft documentation using any of the following methods:

Internet

From the main PeopleSoft Internet site, go to the Documentation section of Customer Connection. You can find order information under the Ordering PeopleBooks topic. Use a Customer Connection ID, credit card, or purchase order to place your order.

PeopleSoft Internet site: <http://www.peoplesoft.com/>.

Telephone

Contact Consolidated Publishing Incorporated (CPI) at
800 888 3559.

Email

Email CPI at callcenter@conpub.com.

Typographical Conventions and Visual Cues

To help you locate and interpret information, we use a number of standard conventions in our online documentation.

Please take a moment to review the following typographical cues:

monospace font

Indicates PeopleCode.

Bold

Indicates field names and other page elements, such as buttons and group box labels, when these elements are documented below the page on which they appear. When we refer to these elements elsewhere in the documentation, we set them in Normal style (not in bold).

We also use boldface when we refer to navigational paths, menu names, or process actions (such as **Save** and **Run**).

Italics

Indicates a PeopleSoft or other book-length publication. We also use italics for *emphasis* and to indicate specific field values. When we cite a field value under the page on which it appears, we use this style: *field value*.

We also use italics when we refer to words as words or letters as letters, as in the following: Enter the number 0, not the letter O.

KEY+KEY

Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press W.

Jump links

Indicates a jump (also called a link, hyperlink, or hypertext link). Click a jump to move to the jump destination or referenced section.

Cross-references

The phrase For more information indicates where you can find additional documentation on the topic at hand. We include the navigational path to the referenced topic, separated by colons (:). Capitalized titles in *italics* indicate the title of a PeopleBook; capitalized titles in normal font refer to sections and specific topics within the PeopleBook. Cross-references typically begin with a jump link. Here's an example:

For more information, see [Documentation on CD-ROM](#) in *About These PeopleBooks*: Related Documentation.

• Topic list

Contains jump links to all the topics in the section. Note that these correspond to the heading levels you'll find in the Contents window.



Name of Page or
Dialog Box

Opens a pop-up window that contains the named page or dialog box. Click the icon to display the image. Some screen shots may also appear inline (directly in the text).



Text in this bar indicates information that you should pay particular attention to as you work with your PeopleSoft system. If the note is preceded by **Important!**, the note is crucial and includes information that concerns what you need to do for the system to function properly.



Text in this bar indicates For more information cross-references to related or additional information.



Text within this bar indicates a crucial configuration consideration. Pay very close attention to these warning messages.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like changed about our documentation, PeopleBooks, and other PeopleSoft reference and training materials. Please send your suggestions to:

PeopleTools Product Documentation Manager
PeopleSoft, Inc.
4460 Hacienda Drive
Pleasanton, CA 94588

Or send comments by email to the authors of the PeopleSoft documentation at:

C:\User\Documentum\Export\DOC@PEOPLESOFT.COM

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions. We are always improving our product communications for you.

CHAPTER 1

Component Interface

Introduction

Every organization depends on real-world business objects—such as invoices and inventory items—to conduct its business. In PeopleSoft applications, *components* represent real-world business objects. For example, an invoice component is a way to capture, store, and display all the essential information related to any given invoice—the general billing and shipping information, plus details about each line item.

Components have *keys* that enable navigation to a specific instance of a business object, and also includes the essential information that describes the object (the fields in the component). Additionally, a component includes an organization's business rules associated with whatever type of business object the component represents.

While online, a user can view, enter, and manipulate data about a business object through the use of a component and its associated pages.

A component interface is a PeopleTools object that you create in Application Designer. It exposes a PeopleSoft component for synchronous access from another application. External applications need not be concerned with the details of page structures and component definitions in order to access the underlying data and business logic through component interfaces. PeopleSoft components can be accessed from the following applications:

- Microsoft's Component Object Model (COM)
- C/C++ shared libraries
- Java
- PeopleCode

An *instance* of a component interface refers to the object at runtime, populated with a single group of data that describes a unique business object. In other words, a *component interface* refers to a type of business object, such as an invoice, while a *component interface instance* refers to a unique version of that business object, such as invoice number 945 versus invoice number 946, and so on.



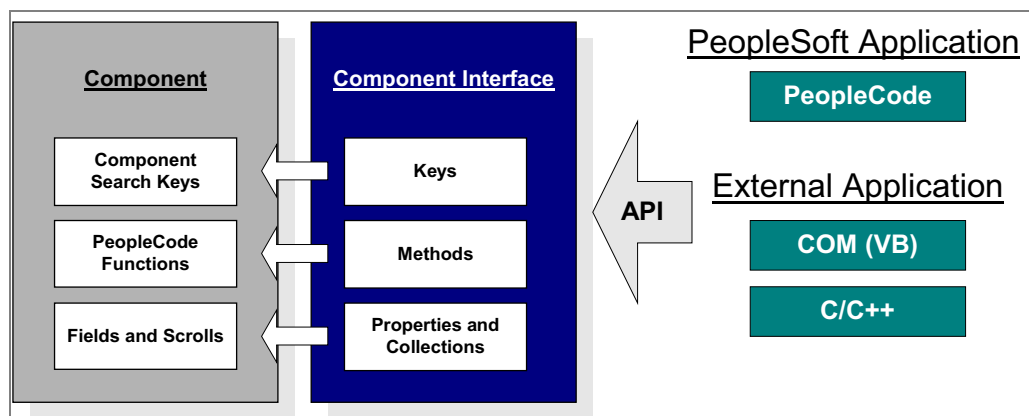
Note: In most cases, component interfaces behave exactly the same as their online counterparts (their associated components). This means that PeopleCode events typically fire in the same order as the online case, and so on. However, there are Comparing Component Interface and Components to this behavior that relate both to PeopleCode processing and search dialog processing.

Component Interface Architecture

The overall component interface architecture includes more than just component interfaces themselves. There are three fundamental elements to the overall component interface architecture—components, component interfaces, and the component interface API.

<i>Elements</i>	<i>Description</i>
Components	One or more pages performing a business transaction that a component interface is associated with.
Component Interface	Exposed aspect of a component. However, unlike components, component interfaces are readily accessible by internal and external applications and multiple component interfaces can reference the same component.
Component Interface API	Application programming interface for a Microsoft COM (Visual Basic) application.
	PeopleCode

The following illustration shows the relationship of the basic elements of the component interface architecture.



Component Interface Architecture

Attributes of a Component Interface

Every component interface has the following four main attributes:

- Component Interface Name
- Keys
- Properties and Collections
- Methods

Component Interface Name

As with every other object in PeopleTools, component interfaces must have a name. The component interface name is used to access it, and should somehow identify the business object that it describes, such as *LOCATION*.

The naming of component interfaces should be consistent and systematic. Also, the name should not be changed once 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 will no longer be able to access it correctly, create a new component interface rather than updating the existing one. There is no “version” property on a component interface, so if you need to create a new version of a delivered component interface, adhere to a standard naming guideline to avoid confusion. A suggested naming guideline is as follows:

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

Keys

Keys define the values that uniquely identify an instance of a component interface. When you create a new component interface, component interface keys are created automatically based on the associated component’s search record. However, you can add or change certain keys, if desired.

A component interface can have three types of keys:

Key Type	Key Characteristics
Get Keys	These keys automatically map to fields marked as <i>Srch</i> in the component’s search record. You need to change Get keys only if you modify the keys of the underlying component after you’ve created a component interface.

Key Type	Key Characteristics
Create Keys	These keys get created automatically if the Use tab on the Component Properties dialog allows the <i>Add</i> action, then Create keys are generated for the component interface automatically. If the component has an <i>Add</i> mode search record, then the component interface uses that search record for the Create keys. Otherwise, the search record is used to generate the keys.
Find Keys	These map to fields marked as both <i>Alt</i> and <i>Srch</i> in the component search record. You may remove Find Keys that you do not wish to make available for searching.



Note. Application Designer automatically creates certain component interface keys based on how some options are set in the component properties, in addition to some of the field options (*Alt* and *Srch*) referenced by the search record.

Properties and Collections

Properties are the individual data items (fields) that describe a component interface. Each property maps to a single field in the component interface's underlying component. A *collection* is a type of property—which points to a scroll, instead of mapping to an individual field, it points to a scroll.



Note: The first item in a component interface collection is always referred to as *item one*, not *item zero*, which is consistent with other PeopleCode processing.

There are two main types of properties: user-defined properties and Standard Properties.

User-Defined Properties

User-defined properties come from a component interface's associated component, and must be added manually. They are the specific record fields that you choose to expose to an external system with the component interface.

Standard Properties

Standard properties are common across all component interfaces and are assigned automatically when a component interface is created. Standard properties also exist for each collection within a component interface. The following table lists the standard properties, including collection and DataRow types. The Application Designer does not display these properties.

Type	Name	What it does...
Standard	CreateKeyInfoCollection	Returns a set of items that describes the Create keys.
	GetKeyInfoCollection	Returns a set of items that describes the Get keys.
	FindKeyInfoCollection	Returns a set of items that describes the Find keys.
	PropertyInfoCollection	Returns a set of items that describes properties.
	GetHistoryItems	Controls whether the component interface runs in “Update/Display” mode or “Correction” mode. Applies only to <i>getting</i> a component interface, not to <i>creating</i> a component interface.
	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. Interactive mode is recommended for most cases where you use a component interface to establish “real-time” integration with another interactive application. However, if you are using a particular component interface as part of a batch process in which thousands of rows are to be inserted, performance may be improved by <i>not running</i> in interactive mode.
	ComponentName	Returns the name of the component class as named in Application Designer
Collection	Count	Returns the number of items in a collection
DataRow	ItemNum	Returns the position of the row within the collection of a DataRow.



For more information on properties, including PropertyInfo properties and related PeopleCode, see Component Interface Classes in the PeopleCode Reference.

Security for Properties

In Application Designer, you control access to user-defined properties by *not* including the property in the component definition or by making the property read-only. This is a global

setting, not related to any individual class or operator ID. PeopleSoft row-level security governs which data values appear for a given property.



For more information on setting up component interface security, see *Setting Component Interface Security*.

Methods

A *method* is an object that performs a very specific function on a component interface at runtime. For each component interface, numerous methods are available. For example, if you are working with a purchase order component interface, you may use a method to approve a specific purchase order. Likewise, you can use methods to save or create a new purchase order. As with component interface properties, there are two main types of methods: User-Defined Methods and Standard Methods.

User-Defined Methods

User-defined methods are those that you can create to meet the requirements of an individual component interface. A method is simply a PeopleCode function that you wish to make accessible through the component interface. Each method maps to a single PeopleCode function.

Standard Methods

Standard methods are those that are available on all component interfaces. They are automatically generated upon the creation of a new component interface in Application Designer, and provide the basic functions required of any component interface.

As with standard properties, standard methods exist for every component interface, as well as for each collection within a component interface. The following are standard methods for **component interface**:

Standard Methods	Action
Cancel	Backs out of the current component interface, canceling any changes made since the last save. Equivalent to clicking the Cancel button online. Returns “True” on success, and “False” on failure.
Create	Creates a new instance of a component interface. Equivalent to opening a new record in <i>Add</i> mode online. Returns “True” on success, and “False” on failure.
Find	Performs a partial key search for a particular instance of a component interface. Returns “True” on success, and “False” on failure.

Standard Methods	Action
Get	Retrieves a particular instance of a component interface. 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.
Save	Saves an instance of a component interface. Equivalent to File, Save in the online system. Returns “True” on success, and “False” on failure.
GetPropertyByName (PropertyName)	Returns the value of a property specified by name. This function typically is used only in applications that cannot get the names of the component interface properties until runtime.
SetPropertyByName (PropertyName, PropertyValue)	Sets the value of a property specified by name. This function typically is used only in applications that cannot set the names of the component interface properties until runtime.
GetPropertyInfoByName (PropertyName)	Returns the information about a property which 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.
CopyRowset (from PeopleCode only)	Enables you to copy rowsets created from the message data in your component interface.
CopyRowsetDelta (from PeopleCode only)	Enables you to copy <i>only the changes</i> created from the message data in your component interface.
Item(Index) Collection method	Takes an item number as a parameter and returns an object of the type 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 object, and so on.

Data Collection Methods	Action
InsertItem(Index)	Inserts a new item. Equivalent to pressing F7 to insert a new row when online. It takes the item number as a parameter, and follows the same conventions for executing business rules (PeopleCode) as the online system.
DeleteItem(Index)	Deletes an item. Equivalent to pressing F8 when online.
Item(Index)	Takes an item number as a parameter, and returns the specified row in the collection.
ItemByKey(keys)	Identifies and finds a specific item based on keys. The keys will vary according to the design of the collection. For more information on determining the key signature, see Getting the Signature of the ItemByKey Method.

CurrentItem:	Returns the current effective DataRow in the collection. Its behavior is consistent with effective date rules used online. This method works with effective-dated records only.
CurrentItemNum:	Returns the item number of the current effective DataRow in the collection. Its behavior is consistent with effective date rules used online. This method works with effective-dated records only.
GetEffectiveItem(DateString, SeqNum):	Returns a pointer to the DataRow that would be effective for the specified date and sequence number. A more general case of the GetCurrentItem function, which returns the object that is effective at this moment. This method works with effective-dated records only.
GetEffectiveItemNum(DateString, SeqNum):	Returns the item number within the collection of the DataRow that would be effective for the specified date and sequence number. A more general case of the GetCurrentItemNum function, which returns the number of the object that is effective at this moment. This method works with effective-dated records only.

DataRow Methods	Action
GetPropertyByName(PropertyName):	Returns the value of a property specified by name. This function typically is used only in applications that cannot get the names of the component interface properties until runtime.
SetPropertyByName(PropertyName, PropertyValue):	Sets the value of a property specified by name. This function typically is used only in applications that cannot set the names of the component interface properties until runtime.
GetPropertyInfoByName(PropertyName):	Returns a PropertyInfo object with the information about 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.

Security for Methods

The following methods provide techniques for accessing component interface properties and property information. Because properties cannot be individually secured within a component interface, these particular methods also cannot be individually secured.

- GetPropertyByName
- SetPropertyByName

- `GetPropertyInfoByName`
- `CopyRowset`
- `CopyRowsetDelta`



For more information on setting component interface security, see [Setting Component Interface Security](#).

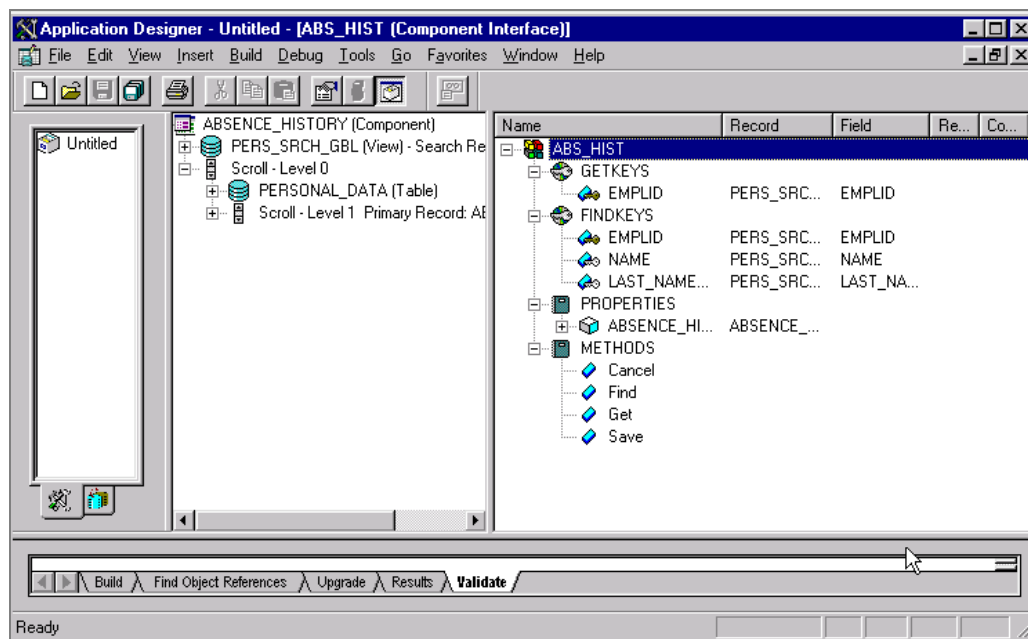
CHAPTER 2

Creating a Component Interface

You create and modify component interfaces using Application Designer. This section assumes that you are already familiar with Application Designer.

Views in Application Designer

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












The Component View and Component Interface View

The component view shows exactly the same hierarchical record structure that you would see if you had the component open in Application Designer. The Component Interface View shows a similar structure.

In general, you add individual objects, or groups of objects, to the component interface by dragging objects from the component view into the component interface view. All objects in the component view are part of the underlying component interface, and they are accessible through user-defined methods or through PeopleCode events on the component. However, only the objects in the component interface view will be exposed to the calling program at runtime.

Component Interface View

The component interface view displays a tree in which each object type is represented by a unique icon. Some icons are used in both the component view and the component interface view with slightly different meanings. The tables below explain the meaning of each icon in the component interface view.

Icon	Description
	Component Interface
	Group of keys
	Property that is a key field from the underlying record
	Alternate search key
	Group of properties or methods
	Property or method
	Collection
	Property that is a required field for the underlying record
	Identifies an item in a component interface that is no longer “in sync” with the underlying component. For example, if a field on which a property depended is deleted from the component, this icon appears.

Component Interface View Display

Columns	What displays...
Name	Name of a specific element of a component interface (such as the name of a property or method).
Record	Name of the underlying record upon which a specific element is based. Note that if this underlying record name changes, the component interface will continue to point to the appropriate record.
Field	Name of the field to which a component interface property points. As with the record name, the underlying field name can change, and the component interface will continue to point to the appropriate field.
Read Only (Y/N)	Displays whether a specific property or collection has been marked read-only.
Comment	Displays any comments that exist in the Edit Property dialog for the selected key, property, or collection.

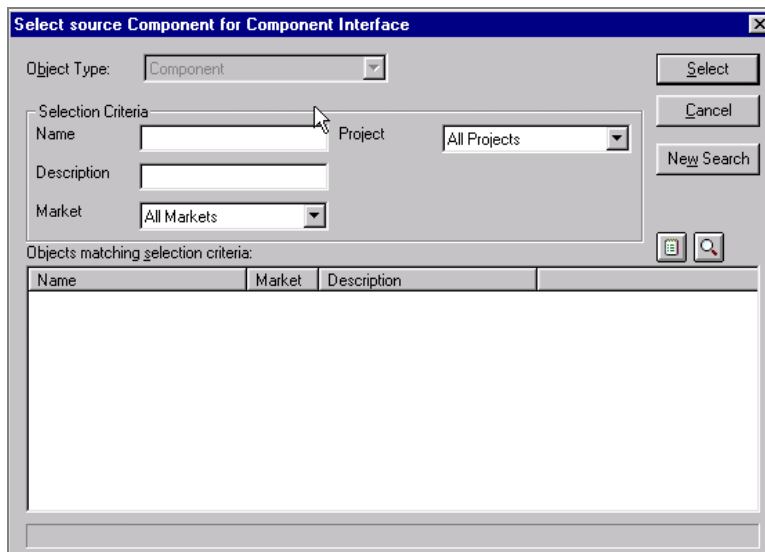
In the component interface view, properties display in the same order as they appear in the component; that is, they are not sorted alphabetically.

Creating a New Component Interface

Because each component interface points to a single component, you must know for which component you are constructing a component interface. You may choose to use an existing component within your 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.

To create a new component interface

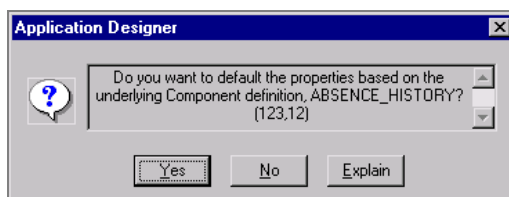
1. Select **File, New** from the Application Designer menu.
2. Select the ***Component Interface*** object type from the **New** dialog.



Selecting a Component for Component Interface

3. Select the component on which this component interface will be based.

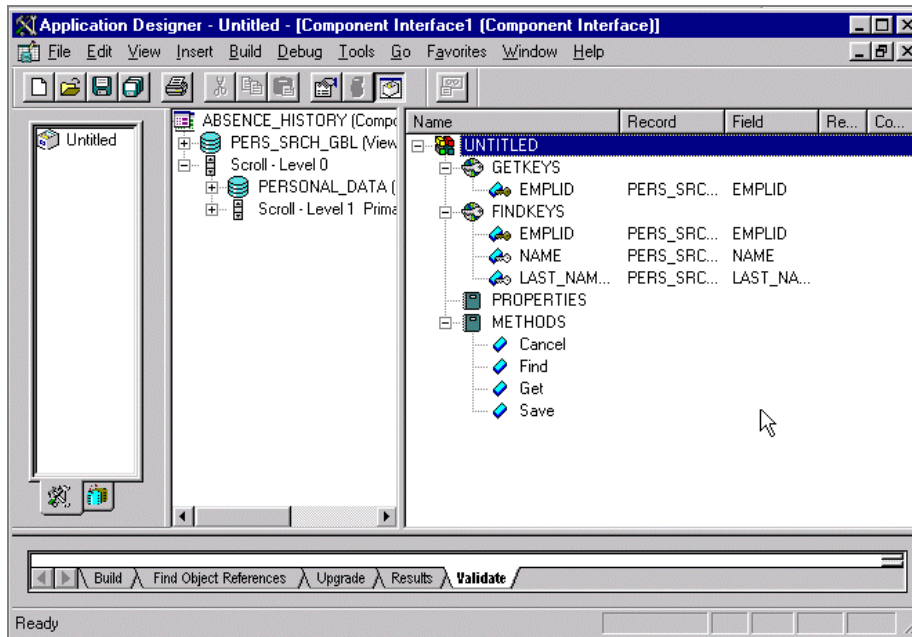
Once you select the appropriate component, you'll see a message asking if you want default component interface properties to be defined based on the fields of the selected component.



Confirming default property values

4. Click **Yes** to confirm the default property definitions, or **No** if you don't want any default properties.

An untitled component interface displays showing the Get keys and Find keys. However, *Create* keys are produced only if the search record of the underlying component is set to run in *Add* mode (the example shown below does not have Create keys, because the search record of the underlying component cannot run in *Add* mode). Application Designer creates the keys for you as you drag and drop objects.



A New Component Interface



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

5. Save the component interface.

When you save a new component interface, Application Designer automatically creates the standard methods Cancel, Find, Get, and Save. Create is not generated automatically unless the component supports the *Add* mode. Therefore, the Create standard method has not been generated for the component interface displayed.

Once you have saved the component interface, you can add user-defined methods to it.

6. Add properties, collections, or methods to the component interface.



For more information on creating properties, collections, and methods, see [Creating Properties](#), [Creating Collections](#), and [Working with Methods](#).

7. Set the security.



For more information on enabling security, Setting Component Interface Security.

8. Test the component interface.

Application Designer includes a helpful feature for testing any component interface you create.



For more information on testing your component interface, see Testing a Component Interface.

Creating Properties

To create a property

1. Drag a record, field, or scroll from the component view to the component interface view.

It does not matter *exactly where* you drop the object in the component interface view. 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 an object from the component view into the component interface view, all “child” objects are brought into the component interface automatically. Once these child properties have been added to the component interface, you can remove each property individually, if necessary.

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

To delete a property

1. Select the property and press the Delete key on your keyboard.

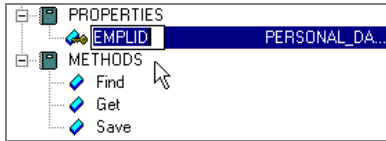
You can also right-click on the property and select **Delete** from the pop-up menu, or highlight the property and select **Edit, Delete** from the Application Designer menu. Standard Windows behavior is employed for selecting multiple properties. That is, you can Shift+click to select a series of properties or Control+click to select multiple, individual properties.

Property names are automatically named according to the corresponding fields from the component. However, it's easy to rename a property if necessary. A renamed property still references the original field, regardless of the name change.

To rename a property

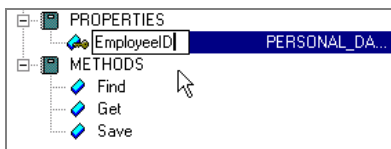
1. Click the property name twice in the component interface view.

Use two “slow” clicks, as opposed to double-clicking. You can also right-click on the property and select **Edit Name** from the pop-up menu, or select **Edit, Edit Name** from the Application Designer menu. In the example below, we’re changing the *EMPLID* property name to *EmployeeID*.



2. Type in the new property name.

Programs accessing this component interface must reference the new property name.



Making Properties Read-Only

You can make any property (including collections) read-only. At runtime, a read-only property can be read, but not updated.

To make a property read-only

1. Highlight the property and select **Edit, Toggle Read Only Access** from the Application Designer menu.

You can also right-click on the property in the component interface view, and select **Toggle Read Only Access** from the pop-up menu. A “Y” appears in the **Read Only (Y/N)** column of the component interface view corresponding to each property that you have selected to be read-only.

Creating Collections

A collection is a property that points to a scroll, rather than a field, in a component interface’s underlying component. Creating collections is similar to creating other properties—you drag the scroll from the component view into the component interface view. There are some important points to keep in mind when creating scrolls, as follows:

- **When dragging a scroll into the component interface view, all “child” scrolls come with it.** This is the same behavior you would expect when creating any 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.
- **Keys that appear in parent and child scrolls are not added to child collections.** In order for the component interface to function as expected, the keys must remain synchronized at

different levels of the component. Having keys at lower levels, makes it possible to compromise this synchronization. Therefore, lower-level keys are not introduced into the component interface.

- **When dragging 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 for you automatically in the component interface. This hierarchy of collections is necessary so that it's possible to navigate to the child collection at runtime.

Adding and Removing Keys

Application Designer makes keys automatically when you create a component interface. Typically, you will have to add keys only if new keys are added to the underlying component after the creation of the component interface. However, you may 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.

To add a key

1. Drag the desired key from the component view to the component interface view.

You first will need to expand the Search key collection (the first collection) in the component view, and then drag the desired key to the component interface view.

To delete a Find key

1. Select the desired Find key in the component interface view and press the Delete key.

Which Properties to Expose?

You easily create component interface by dragging a scroll from the component view into the component interface view. However, some forethought is required before exposing a component as a component interface. Certain components, in fact, must be carefully exposed to ensure that they behave as you would expect.

Guidelines for Exposing Components

The first time you drag a scroll from the component view to the component interface view, the system follows certain rules to determine what properties to expose.

- **Considerations about levels.** Keys are exposed only at the highest level collection in which they first appear. In some cases, this is not desirable. When an effective-dated page that has the same level zero and level one record is exposed through a component interface, it should be exposed in exactly the same way it is displayed on the page. In this case, only one key field typically appears at level zero and the effective-date keys appear at level one. Your component interface wrapper should expose the page in the same fashion—removing keys that do not

appear on the level zero scroll in the page from the component interface top-level collection, and manually adding those keys that appear on level one scroll in the page to the second-level collection.

Typically, you will not want to expose Get or Create keys since these are set before a Get or Create operation and might be inadvertently changed.

- **No Add mode on page.** If your page does not support *Add* mode, then typically you will not want to expose the level zero record of the component, as it will contain data that is not specific to the component interface you are creating.
- **Invisible fields.** You should not expose fields that are not visible in the component view. The component optimization code may eliminate unused fields from its buffers in which case an error will result when that field is accessed by the component interface.

Working with Methods

A *method* is an object that performs a specific function on a component interface at runtime. Each method is simply a PeopleCode function made accessible to other programs. As with properties, methods are saved as part of a component interface definition. There are two main types of methods: *standard methods* and *user-defined methods*.



For more information on PeopleCode related to component interface, see Component Interface Classes in the PeopleCode Reference.

Standard Methods

By default, each component interface is created with the standard methods—Cancel, Find, Get, Save—enabled. Additionally, the Create standard method is generated if *Create* keys have been added to the component interface. When creating a new component interface, you must save the component interface before the standard methods will be created. Application Designer adds the standard methods upon the first save of a new component interface.

You can control whether or not standard methods are accessible at runtime. Follow the procedure below to enable or disable any standard method.

To enable or disable standard methods

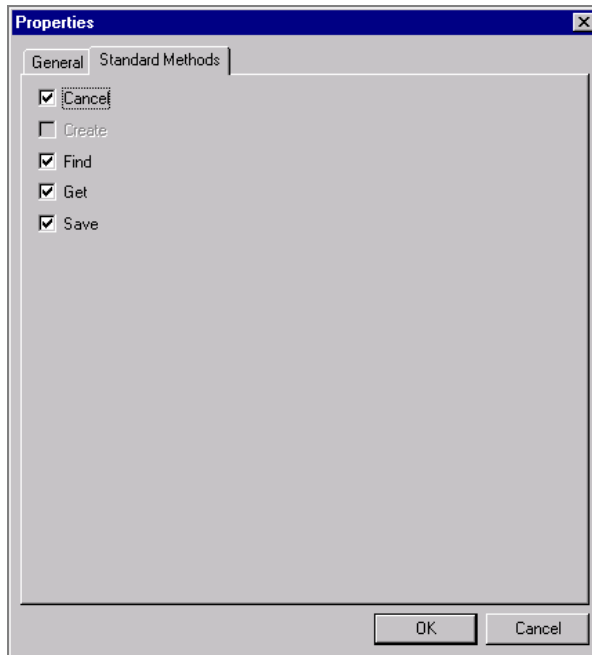
1. Select **File, Object Properties** from the Application Designer menu.

You can also right-click anywhere in the component interface view and select **Component Interface Properties** from the pop-up menu. The **Object Properties** dialog opens.

2. Click the Standard Methods tab.

You can enable or disable any of the standard methods selecting the corresponding checkbox. Doing so determines whether or not the method is available at runtime when the component

interface is accessed. Create is grayed out in the example below. This is because no Create keys exist for this component interface, which indicates that the search record for the underlying component cannot run in *Add* mode.



Enabling Standard Methods

User-Defined Methods

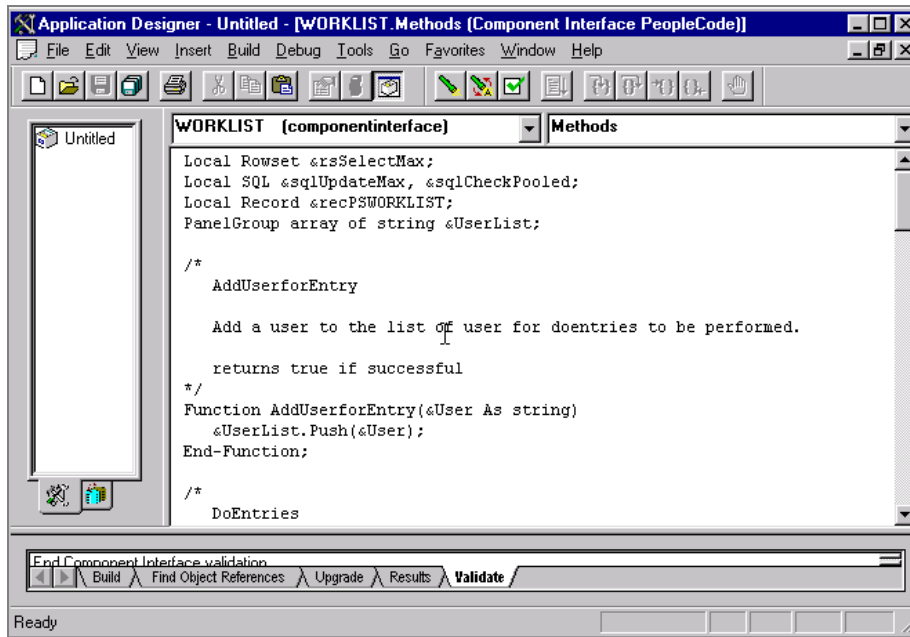
To create a user-defined method

1. Right-click anywhere in the component interface view, and select **View PeopleCode** from the pop-up menu.

You can also highlight any object in the component interface view, and then select **View, View PeopleCode** from the Application Designer menu. The PeopleCode editor appears. With a new component interface, initially there will be no PeopleCode displayed in the editor, because no user-defined methods have been created yet.

2. Write the required PeopleCode functions.

Any PeopleCode functions you write will be stored in a single PeopleCode program attached to the component interface. You must set permissions for every user-defined method. If you've set permission to *Full Access*, at runtime, that function for the component interface will be exposed to calling programs as a method.



Creating User-Defined Methods



New user-defined methods do not appear in the list of methods until you save the component interface.

Setting Component Interface Security

After creating a component interface, you need to set security for it before the component interface can be tested or accessed. As with other PeopleTools objects, access must specifically be granted before a component interface is available for use at runtime by any user. Additionally, before a component interface can be tested, security access must be given to the appropriate class, so that the desired user(s) can access the component interface.

There are essentially two ways to secure component interfaces:

- Use Maintain Security to set security. Maintain Security addresses component interface security in the same manner that it addresses security for other PeopleTools objects. You can use it to control access to individual methods or entire component interfaces.
- Use Application Designer to mark individual properties “read only.” Any property can be marked “read only” in the component interface design. For more information, see Making Properties Read-Only.

To set up component interface security

1. From your browser, select PeopleTools, Maintain Security, Use, Permission Lists.
2. Select the permission list to which you want to set security.

Permission Lists

Find an Existing Value

Permission List:

[Basic Search](#)

[Add a New Value](#)

Search Results

View All First 1-21 of 21 Last

Permission List	Permission List Description
ALLPANLS	(blank)
APPSRVR	Can start application server
BENADMIN	Benefits Administrator
CCADMIN	(blank)
CCDVLP	(blank)
CCREST	(blank)
CUSTOMER	Customer Class
EMPLOYEE	Employee Class
HRPNLS	(blank)
INPNLS	(blank)
INVPANLS	(blank)
POBUYER	Purchasing Buyer
PRODREP	Production Representative
PTMSG	(blank)

3. Select the Component Interface tab.

General Pages PeopleTools Process Sign-on Times **Component Interface** Message Monitor

Permission List: ALLPANLS

Description:

Permission List General

Navigator Homepage:

☐ Can Start Application Server?

☐ Allow Password to be Emailed?

Time-out Minutes

☒ Never Time-out

☐ Specific Time-out (minutes)

Component Interface tab in Maintain Security

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

General Pages **PeopleTools** Process Sign-on Times

Permission List: ALLPANLS


Description:

Edit	Name		
Edit	ABS_HIST	+	-
Edit	BUS_EXP	+	-
Edit	DISCIPLN_ACTN	+	-
Edit	NVSRPTS	+	-
Edit	PROCESSREQUEST	+	-
Edit	PSACTIVITYLOG	+	-
Edit	USER_PROFILE	+	-

View All First 1-7 of 9 Last

Save Return to Search Previous tab Next tab Add Update/Display

Component Interface list

If you want to add another component interface to the list, click . Enter the component interface name in the text box.

Edit	Name		
Edit	ABS_HIST	+	-
Edit	<input type="text"/>	+	-
Edit	BUS_EXP	+	-
Edit	DISCIPLN_ACTN	+	-
Edit	NVSRPTS	+	-
Edit	PROCESSREQUEST	+	-
Edit	PSACTIVITYLOG	+	-

View All First 1-7 of 10 Last

Inserting a Component Interface

The **Authorized Component Interface** screen appears, showing all methods (both standard and user-defined) within the component interface and their method access.

Component Interface Permissions

Authorized Component Interface

ABS_HIST

View All First 1-4 of 4 Last

Method	Method Access
Cancel	FULL
Find	FULL
Get	FULL
Save	FULL

Full Access (All)

No Access (All)

OK Cancel

Setting Access Permissions for Methods

5. Set the **Access Permission** for each method.

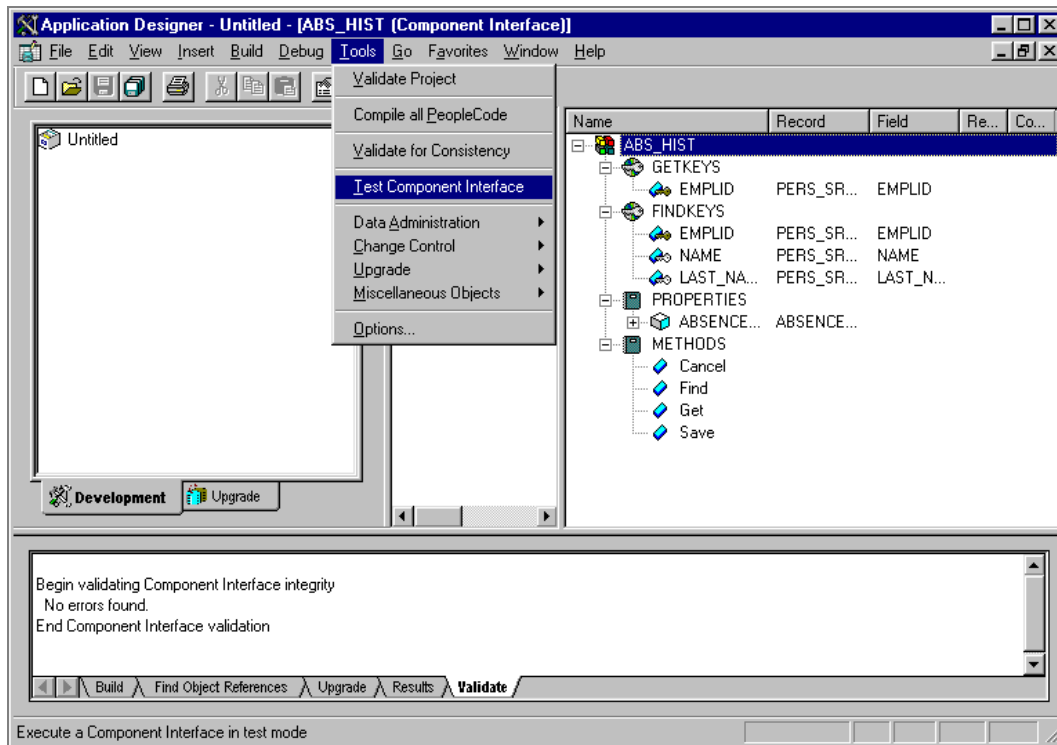
For each method within the component interface, you can choose between **Full Access** and **No Access**. You must grant **Full Access** to at least one method to make the component interface available for testing and other online use. Click **OK** when done.

Testing a Component Interface

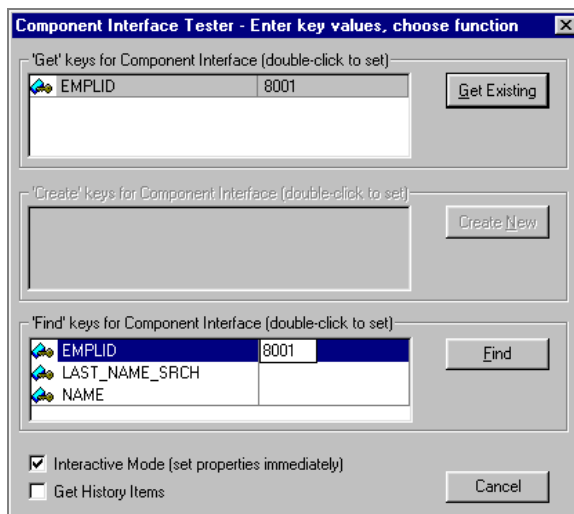
After setting the security parameters for a component interface, you can test the contents and behavior of that component. When you are working with a test component, real data from the database is used. Therefore, if you save the information you change by calling the Save method, it will be changed in the database.

To test a component interface

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



You can also right-click anywhere in the component interface view and select **Test Component Interface**. The **Test Component** dialog appears. This dialog displays the key structures (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.



Testing a Component Interface

3. Enter key values.

To enter a key value, double-click in the column to the right of any displayed keys. You can then edit the value in the right-hand column. The data used for the test will correspond to the key values you enter here. In the example above, we've entered an employee ID of 8001.

4. Select whether to run in **Interactive Mode**.

If you select the **Interactive Mode** box, this means that the component will be sending each "set property" request to the application server immediately, instead of storing them up to be sent in batches; it means that edit processing (and other processing, such as FieldChange PeopleCode) will occur for each transaction.

Whether or not you select this option depends on how you expect a particular component interface to be used, as well as what you are specifically testing at the moment. In a real production system, this parameter can significantly affect performance, but it makes little difference in the test component. In non-interactive mode, errors and properties are not updated until a method is executed. By default, **Interactive Mode** is turned on.

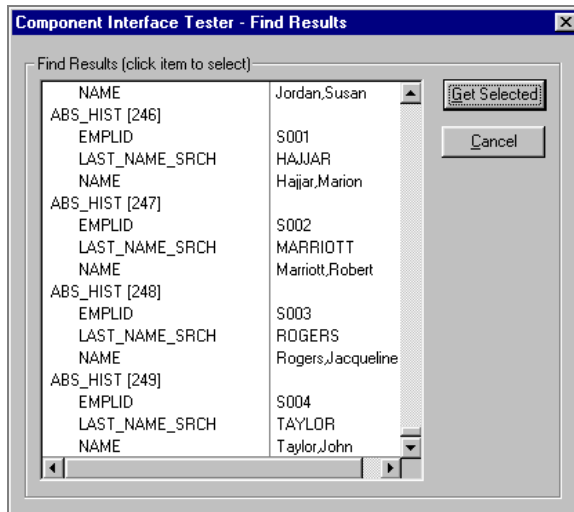
5. Select whether to **Get History Items**.

Selecting this option determines whether to retrieve "history" data. This option applies to effective-dated fields only, and is equivalent to running in either *Update/Display* mode or *Correction* mode online. This option is initially turned off.

6. Decide whether to **Get Existing** records or to **Create New** a new one for the test.

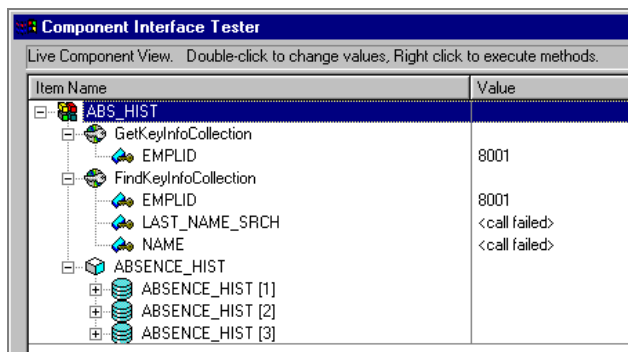
The **Get Existing** option is equivalent to opening a record in *Update/Display* or *Correction* mode online. It tests calling the Get method through the Component Interface API. The **Create New** option is equivalent to creating a new record in *Add* mode online. It tests calling the Create method through the Component Interface API. If your component does not support the Create method, this button will be disabled.

If you want to enter a partial key, use the **Find** option. Application Designer will then use the values in the FindKeyInfoCollection tree to return a set of target components. You then can choose a single 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. This is the same as calling the Find method through the Component Interface API; followed by selecting a value from the Find results, and then setting the Get key and calling the Get method.



Using the Component Interface Tester's "Find" Option

After you click either the **Get Existing**, **Create New**, or **Find** button, the Component Interface Tester dialog appears.



Using the Component Interface Tester

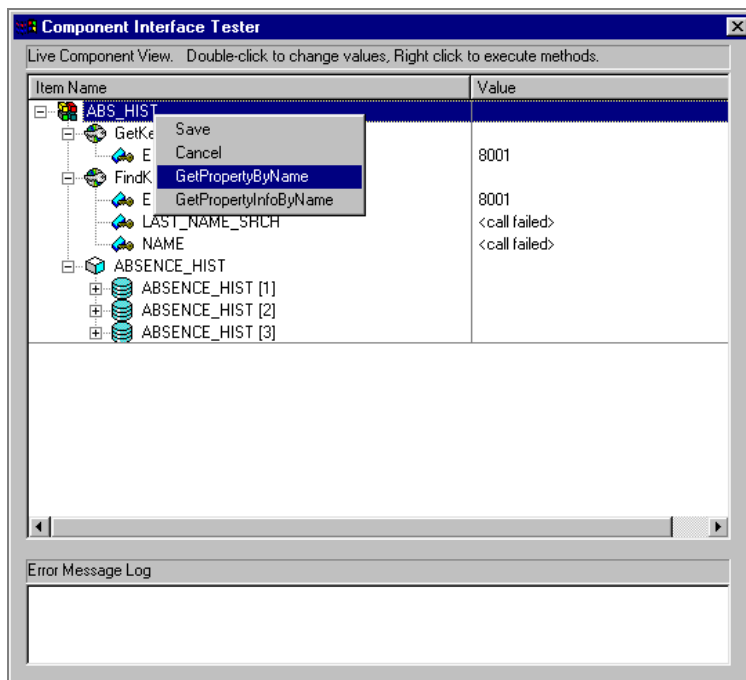
7. Test component interface properties.

To change a value, double-click on a value and enter a new value. Note that the test component interface uses real data. If you save the information you've changed by calling the Save method, the information will be changed in the database (in Interactive Mode). Some basic validation is done when you leave the field—which is equivalent to tabbing off of a field in the online case. This validation includes system edit, FieldChange PeopleCode events, and FieldEdit PeopleCode events. Further validation may be done when the Save method is called (SaveEdit, SavePreChange, Workflow, and SavePostChange). If errors or warnings are encountered, they are displayed in the **Error Message Log** at the bottom of the window. The Error Message Log displays the same text that would appear in the Session object PS-Messages collection if you were accessing the component through the Component Interface API.

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

A pop-up menu appears showing the **Save** and **Cancel** standard methods, plus 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 in which you can enter the parameters will appear. After the method executes, the same dialog appears again, displaying any change to the parameters caused by the method. The return value of the function is displayed in the title of the dialog. If a component interface requires no parameters, you will not see the initial dialog, but will see the return value dialog following the function call.



Testing Component Interface Methods



Because the execution of a component interface method can result in a change to the component interface structure, Application Designer will always redraw the component interface tree in its collapsed form following a method call.

9. Test collection methods by right-clicking on the collection name.

A pop-up menu appears showing the standard collection methods. Select the collection method you want to test for this component interface. After you select a collection method to test, the **Enter parameters** dialog prompts you to enter an item number for the collection method you are testing. The **index [Number]** you enter is used to retrieve, insert, or delete an item, according to the rules discussed below.



Using the Enter Parameters Dialog

After you enter an **index [Number]**, the result is shown in the dialog. If there is a return value, it is displayed in the title bar. Otherwise the message “No value” is displayed. Click **OK** or **Cancel** to dismiss the dialog.

The purpose of each collection method is as follows:

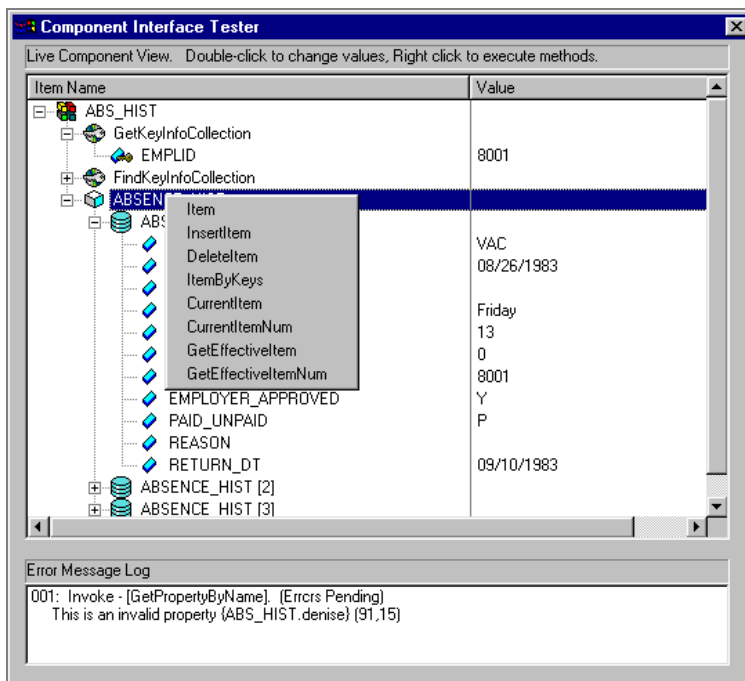
- **Item(index):** Returns the row at the specified index. Only the success or failure of this routine is of interest from within the test component.
- **InsertItem(index):** Inserts a new item. Equivalent to pressing the F7 button online. A new item will be inserted following the **index [Number]** you specified on the **Enter parameters** dialog.
- **DeleteItem:** Deletes the item number you specified on the **Enter parameters** dialog. Equivalent to pressing the F8 button online.
- **ItemByKeys(key1, key2, ...):** Returns the row corresponding to the specified keys. Only the success or failure of this routine is of interest from within the test component.
- **CurrentItem:** This method returns the effective row in an effective-dated record. Only the success or failure of this routine is of interest from within the test component.
- **GetEffectiveItem(DateString, SeqNum):** Returns a pointer to the DataRow that would be effective for the specified date and sequence number. A more general case of the GetCurrentItem function, which returns the object that is effective at this moment. This method works with effective-dated records only.
- **GetEffectiveItemNum(DateString, SeqNum):** Returns the item number within the collection of the DataRow that would be effective for the specified date and sequence number. A more general case of the GetCurrentItemNum function, which returns the number of the object that is effective at this moment. This method works with effective-dated records only.

Getting the Signature of the ItemByKeys Method

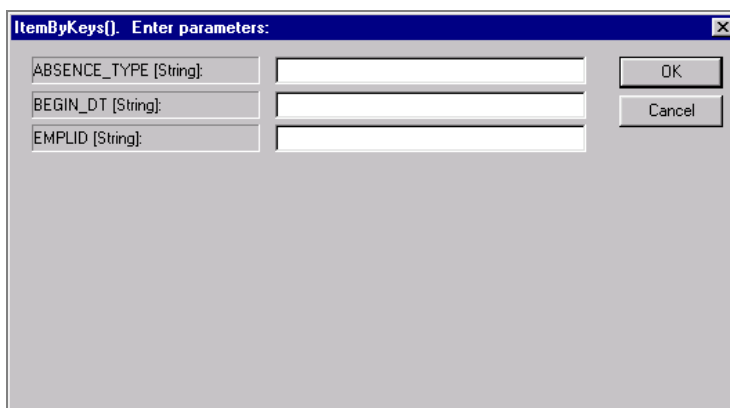
You can get the signature for the ItemByKeys method (or any other method) when testing a component interface. Open the object, and select **Tools, Test Component**. Navigate to the

desired collection, right-click on it, and select **ItemByKeys** from the pop-up menu. A dialog appears showing you the specific parameters, types, and the order in which you should call **ItemByKeys**. This is particularly helpful for the **ItemByKeys** method, because its signature is different for each component interface.

In the following example, the keys for **ABSENCE_HIST**'s **ItemByKeys** method are **ABSENCE_TYPE** (String), **BEGIN_DT** (String), and **EMPLID** (String).




Getting the Signature of the ItemByKeys Method



Viewing the Signature of the ItemByKeys Method

Validating a Component Interface

Validation ensures that a component interface definition has not deviated from its source component. This can happen whenever a component deletes or adds a record or field. It can also happen if the key structure is modified by adding or removing keys. Properties and keys that are no longer synchronized with their associated component are marked with the  icon.



With respect to component interfaces, validation is the process of checking whether the underlying component of a component interface has changed. It does *not* validate the PeopleCode associated with a component interface. To validate the PeopleCode, you must open the component, and then select **Tools, Validate** from the Application Designer menu.

To correct an invalid component interface, you may have to delete properties for which there are no longer appropriate fields or records. If the structure of the source component has changed, you may have to delete old properties and re-add the new properties in their appropriate locations. If a new property provides the same functionality as a previous property, you can change the name of the new component interface property back to its original name, which will make it appear to external applications as though the component interface has not changed. This will work only if the new component interface is not structurally different than the original component interface. That is, the properties still appear at the same collection levels.

To validate a component interface

1. Open the component interface in Application Designer.

Validation occurs automatically whenever you open a component interface in Application Designer.

2. Select **Tools, Validate for Consistency** from the Application Designer menu to validate an open component interface.

You can also right-click anywhere in the component interface view and select **Validate for Consistency**. As you make changes to components, component interfaces, or other related objects, you may want to validate a component interface that you already have open in Application Designer, rather than closing and re-opening the component interface to force validation to occur. Use this feature to validate a currently open component interface.

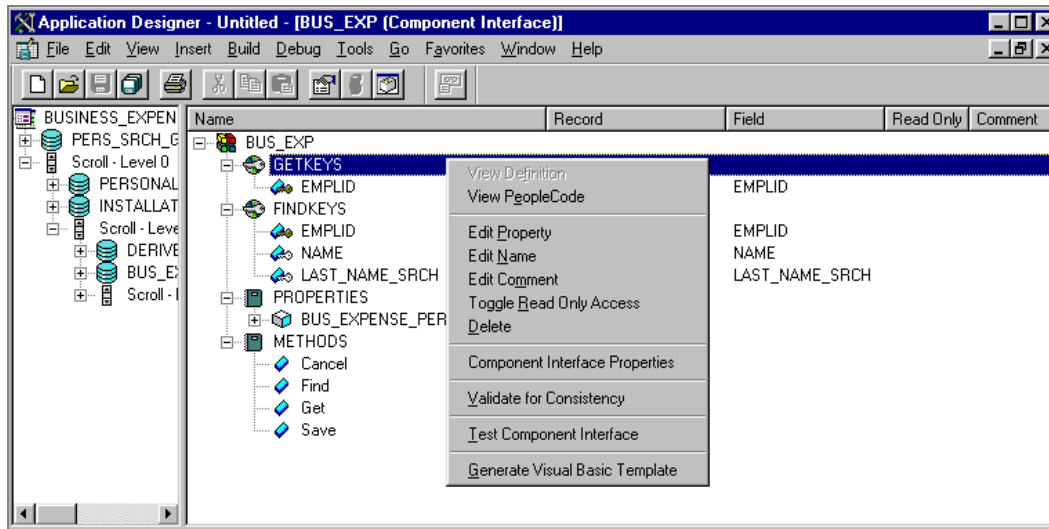
Generating Visual Basic Template

After creating and testing the component interface, you can begin coding the run-time portion of the application. From Application Designer, you can generate a Visual Basic template based on your component interface. Then you can modify the template as needed.

To generate a Visual Basic Template

1. Open a component interface definition in Application Designer.

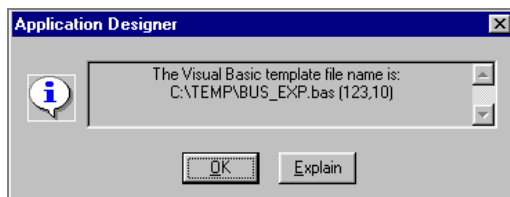
Right-click anywhere in the definition view to display the pop-up menu.



Generate Visual Basic Template pop-up

2. Select Generate Visual Basic Template.

When the template is successfully generated, the following message displays:



Generated VB Template message

3. Open the generated file and modify the source code, as needed.

Example of Visual Basic Generated File

The following file is a dynamically generated Visual Basic template you can use as a sample. You need to replace all <*> notations with valid Visual Basic variables.

```
Private Sub ABS_HIST()
```

```
On Error GoTo eMessage
```

```
'***** Set Object References *****

Dim oCISession As Object

Dim oABS_HIST As Object

Dim oABSENCE_HIST As Object

Dim oABSENCE_HISTItem As Object

'***** Set Connect Parameters *****

strAppSeverPath = <*>

strOperatorID = <*>

strPassword = <*>

'***** Create PeopleSoft Session Object *****

Set oCISession = CreateObject("PeopleSoft.Session")

'***** Connect to the App Sever *****

oCISession.Connect 1, strAppSeverPath, strOperatorID, strPassword, 0

'***** Get the Component *****

Set oABS_HIST = oCISession.GetCompIntfc("ABS_HIST")

'***** Set the Component Interface Mode *****

oABS_HIST.InteractiveMode = False

oABS_HIST.GetHistoryItems = True

'***** Set Component Get/Create Keys *****

oABS_HIST.EMPLID = <*>

'***** Execute Get Or Create *****
```

```

oABS_HIST.Get

'***** BEGIN:  Set Component Interface Properties *****

'***** BEGIN:  Set Component Interface Properties *****

'Set ABSENCE_HIST Collection Field Properties -- Parent: PS_ROOT Collection
Set oABSENCE_HIST = oABS_HIST.ABSENCE_HIST

'For <*> = 1 to oABSENCE_HIST.Count

Set oABSENCE_HISTItem = oABSENCE_HIST.Item(<*>)

oABSENCE_HISTItem.EMPLID = <*>

oABSENCE_HISTItem.ABSENCE_TYPE = <*>

oABSENCE_HISTItem.BEGIN_DT = <*>

oABSENCE_HISTItem.RETURN_DT = <*>

oABSENCE_HISTItem.DURATION_DAYS = <*>

oABSENCE_HISTItem.DURATION_HOURS = <*>

oABSENCE_HISTItem.REASON = <*>

oABSENCE_HISTItem.PAID_UNPAID = <*>

oABSENCE_HISTItem.EMPLOYER_APPROVED = <*>

oABSENCE_HISTItem.COMMENTS = <*>

oABSENCE_HISTItem.DAY_OF_WEEK = <*>

'Next <*>

'***** END:  Set Component Interface Properties *****

'***** END:  Set Component Interface Properties *****

'***** Save Component Interface *****

oABS_HIST.Save

oABS_HIST.Cancel

```

```
Exit Sub

eMessage:

'***** Display VB Runtime Errors *****

MsgBox Err.Description

'***** Display PeopleSoft Error Messages *****

If oCISession.PSMessages.Count > 0 Then

    For i = 1 To oCISession.PSMessages.Count

        MsgBox oCISession.PSMessages.Item(i).Text

    Next i

End If

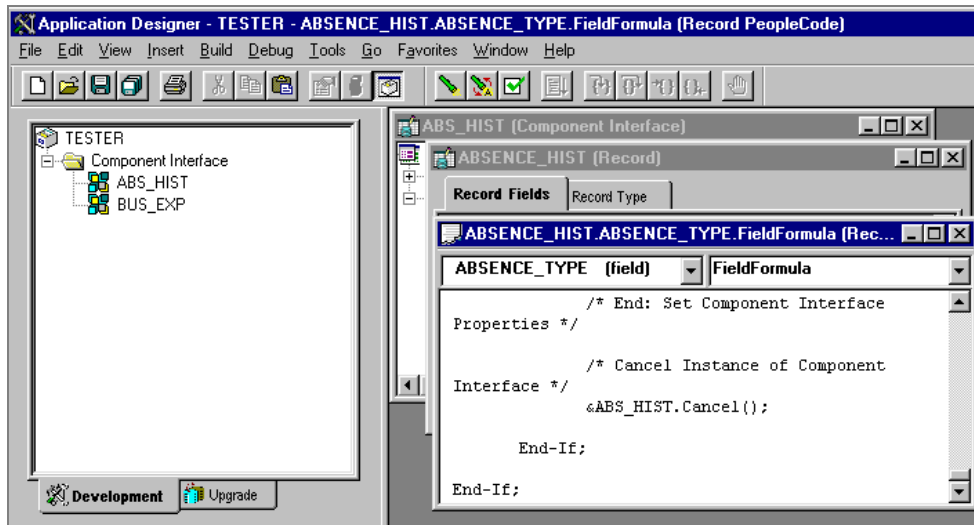
End Sub
```

Generating PeopleCode

After creating and testing the component interface, you can generate the PeopleCode and then modify it, if needed.

To generate PeopleCode from a component interface

1. Open a component interface definition.
2. Insert the component interface into a project.
Select **Insert, Current Object into Project**. Save the project.
3. Open the PeopleCode editor.
4. Select the component interface from the project workspace.
Drag and drop the object from the project into the PeopleCode editor.



PeopleCode generated by dragging and dropping ABS_HIST component interface

5. You can make any necessary changes to the PeopleCode in the PeopleCode editor window.

You must replace the “<*>” notations, which are variable place holders, with specific values for your program before executing the PeopleCode.

CHAPTER 3

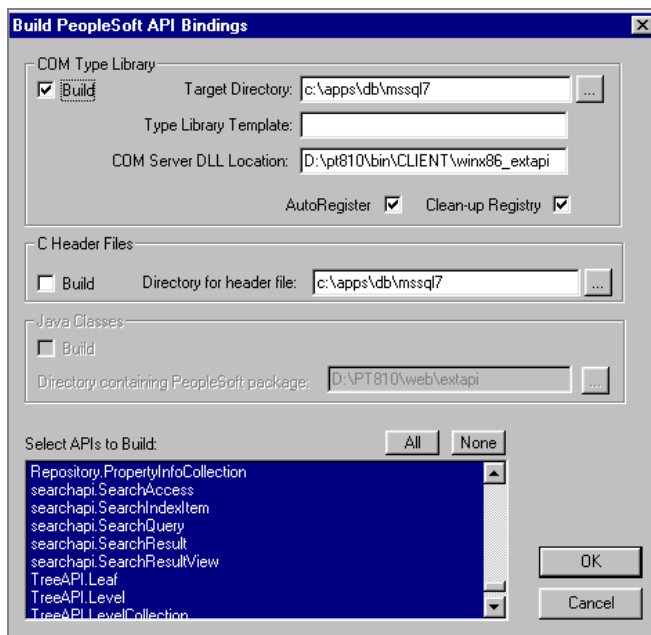
The Component Interface API

After creating your component interface, you need to create an API which will build the dynamic link libraries, classes, and registry settings necessary to allow a third party application to communicate with PeopleSoft. These files reside on the client machine; that is, the web server for ASP, and the machine running the Java program for Java. The registry file may also need to be executed to update the registry with the new libraries.

Only external applications, such as COM or C/C++ programs, require a component interface API. PeopleCode programs do not require a component interface API, and in fact, we **do not** recommend building a Component Interface API if a component interface is to be accessed from PeopleCode only.

To build a component interface

1. Open any component interface.
2. Select **Build, PeopleSoft APIs** from the Application Designer menu.



Build PeopleSoft API Bindings dialog

3. Set options for COM Type Library area.

Build	Select this checkbox if you are building the API so that it can be accessed from Visual Basic programs. Otherwise, the type library will not be built.
Target Directory	This is the directory to which the API is built.
Type Library Template	Specifies the type library template you want to use.
COM Server DLL Location	<p>This identifies the directory in which the PeopleSoft API Adapter (psapiadapter.dll) resides on the workstation. The default location is the PeopleTools “bin” directory (set by the PS_HOME variable set in Configuration Manager).</p> <p>The system creates the following type library files in the Location you specify: <i>Peoplesoft_Peoplesoft.tlb</i> <i>Peoplesoft_Peoplesoft.reg</i></p>
Auto Register	Select this option to create the registry information required for the type library and execute it immediately, so that your workstation’s registry will be immediately updated without having to restart it.
Clean Up Registry	Select this option to set up the clean up registry.



These two files are only on the workstation used to build the API. All other workstations that will make use of the API must have these files copied to the corresponding directories. Then, the registry file (Peoplesoft_Peoplesoft.reg) must be imported to each workstation’s registry by executing (double-clicking) the generated registry file. If the directory structure differs from the original workstation on which the APIs were built, then you must edit the registry file to change the location of the psapiadapter.dll to reflect the correct directory structure, and then import the registry file on the workstation.

4. Set options for C programs as required.

If you are building the API so that it can be accessed from C programs, click the **Build** checkbox that appears in the **C Header Files** frame. Otherwise, the header file(s) will not be built.

The **Location** identifies the directory in which the header files must reside on the workstation. The default location is the directory from which you started PeopleTools.

5. Build the APIs.

Currently you cannot select an individual API to be built. If you create a new API or modify an existing one, you will have to rebuild all the APIs.



Done message...



Note. The directory containing the PeopleSoft API Adapter (psapiadapter.dll) needs to be set in the PATH environment variable for C Header Files.

Binding Considerations

This section describes some things to consider depending on the binding type: COM bindings on a local workstation or web server or C Header file bindings.

COM Binding

When deploying component interface on a local workstation or web server with COM binding, you need the following:

- Third party application (non-PeopleSoft)
- Type library called *PeopleSoft_PeopleSoft.tlb*. This type library is not specific to a single database instance—it is specific to those database objects.
- Registry file called *PeopleSoft_PeopleSoft.reg*. This registry file is not specific to a single database instance. It is specific to the path settings for the typelib and the psapiadapter.dll that you chose during the Build API. Be aware that often the machine you are deploying to is *not* the machine that you ran the Build APIs on.
- External API installation
- PeopleSoft Application Server
- PeopleSoft application

Third Party Application

For applications written in Visual Basic, Excel Visual Basic for Applications (VBA), or other COM languages, note the following:

- If your program is early-binding, there is a direct reference to the path of the typelib in your code. Therefore, as you deploy you must have the typelib in the same directory on each machine.
- If your program is late-binding, there is no a direct reference to the path of the typelib in your

code. Your code will look in the registry for the path to the typelib. Therefore, as you deploy you can have the typelib in different directories on each machine. You do need to update the registry settings as part of the deployment. This is a more flexible approach.

External API Installation

You will be doing the External API installation on each workstation that runs the non-PeopleSoft application.

During the External API install, you will be prompted for the directory that you want the External API files to go. This needs to be the same directory that you used as the settings for the Build APIs process.

Note that the External API requires a Java Virtual Machine because the calls to the application server are done through JOLT because it supports multi-threading.

C Header Binding

When deploying component interface with C Header binding, you need the following:

- Third party application (non-PeopleSoft)
- C Header files: `peoplesoft_peoplesoft_.i.h`
- External API installation
- PeopleSoft Application Server

Third Party Application

For applications written in C or C++, note the following:

- 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 you include the setup of the path to the *psapiadapter.dll*.

C Header File

When you do the Build API process in the Component Interface Designer, it creates one *peoplesoft_peoplesoft_.i.h* file for all of the objects in the PeopleSoft database. This C header file is not specific to a single database instance—it is specific to those database objects.

Connecting to a Component Interface

To access a component interface from a Visual Basic, C/C++ application, or PeopleCode, you must first build the component interface object.



Any workstation, running a Visual Basic program, requiring access to the API must have DLL files, component interface type library and registry information copied to the appropriate locations. For a web application the above mentioned files need to be copied on the Web Server only.

Installing External Client Settings for the API

Before a client machine can access component interfaces with COM, certain environment settings must be set up on the client workstation. These settings are *not* required for PeopleCode access to component interfaces. Each client workstation that accesses component interfaces through an external (non-PeopleCode) application will need the external API directory installed.



For more information about installing the external client settings, consult the *PeopleSoft 8 Installation and Administration Guide* for your database platform.

Comparing Component Interface and Components

In many ways, accessing a component interface is equivalent to working with an online component. However, the fact that component interfaces *are not equivalent to components* means that there are a few key areas in which you'll see differences between component interfaces and components. For example, search dialog processing and some PeopleCode events are different.

Differences in 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.

Differences in PeopleCode Event and Function Behavior

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

- **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 aren't 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.

Limitations of Client-Only PeopleCode

- Component interface can run on either the client or the server. A component interface runs on the client only if *both* of the following conditions are true: (Otherwise, the component interface runs on the server.)
 - The code calling the component interface is running on a client machine.
 - The second parameter of the Connect method is *EXISTING*.
- Component Interfaces must run either entirely on the server or entirely on the client. To ensure this restriction, component interface references declared in PeopleCode must be declared as local 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.



For more information see Client-Only PeopleCode.

Email from a Component Interface

If you want to use a component interface to send email, use TriggerBusinessEvent PeopleCode event, and not SendMail.

WinMessage Unavailable

You cannot use WinMessage in a component that will be used to build a component interface. You should use MsgGet() instead.

Calling another Component Interface

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 should not call itself from a user-defined method.

CHAPTER 4

Component Interface Example

This section describes the steps in creating a sample component interface.

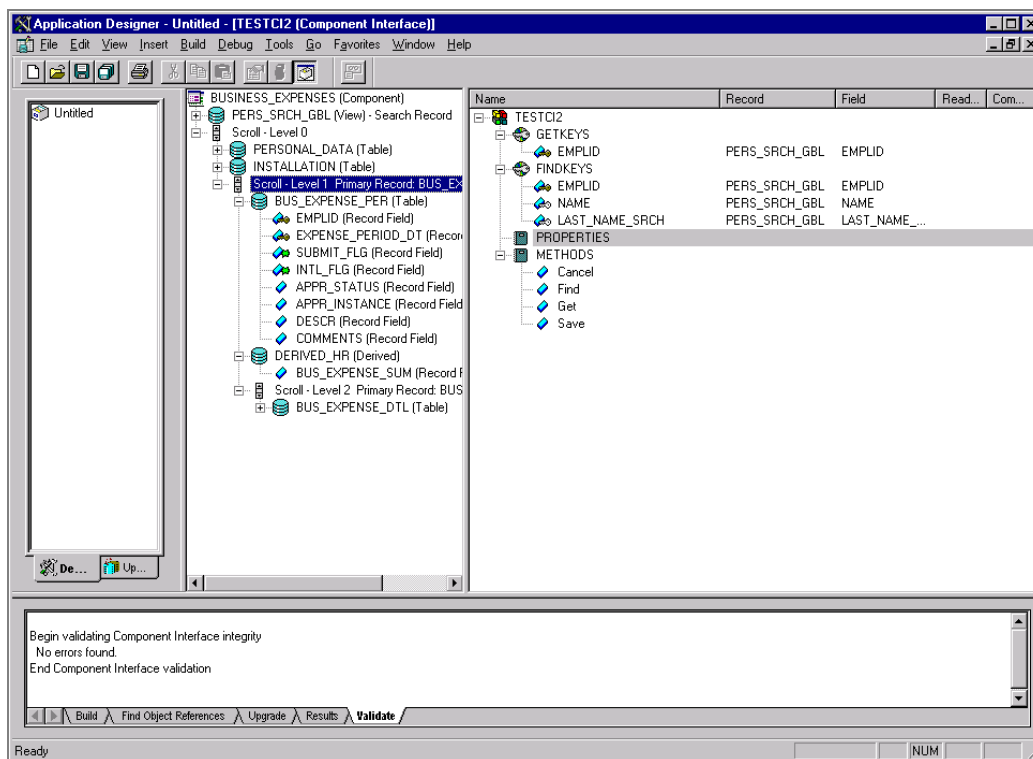
To create a component interface

1. Create a new component interface in Application Designer by selecting **File, New, Component Interface**.

The system prompts you to open the component on which the component interface will be based.

2. Add properties to the component interface by dragging fields, tables, or scrolls over to the rightmost pane.

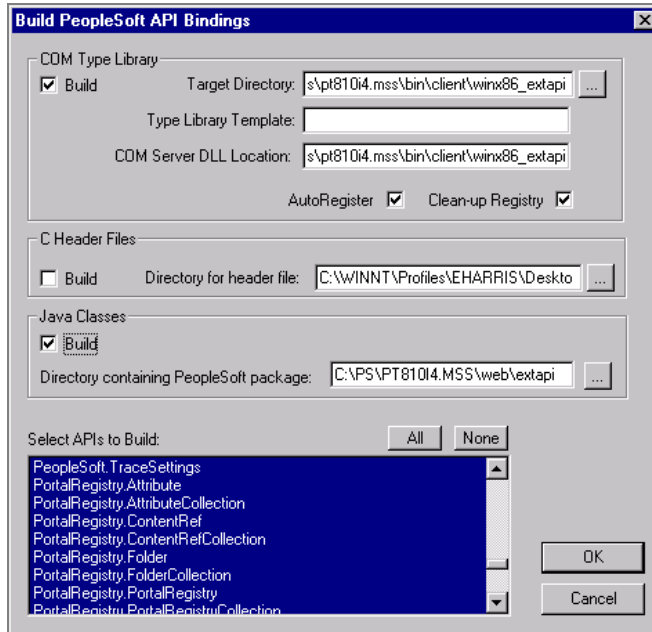
The easiest way to add properties is to drag level 1 scrolls to the right side and drop. This exposes the data that a normal page would have access to in the component.



Creating a component interface

3. Save the component interface with a unique name.

You are ready to build the component interface. Highlight an object in the rightmost pane and select **Build, PeopleSoft APIs**.



Building the APIs

PeopleCode Example

The following example shows how a PeopleCode program might call a component interface named `ABS_HIST`.

```

rem Declare variables;

Local ApiObject &SESSION;

Local ApiObject &CI;

Local ApiObject &ABS_HISTCollection;

Local ApiObject &ABS_HISTItem;

rem Establish a PeopleSoft Session;

rem The Connect method connects a session object to a PeopleSoft application
server;

rem Syntax : (version, {"EXISTING" | ConnectID:Port}, OperatorID, Password,
ExtAuth);

```

```

&SESSION = GetSession();

If (&SESSION.connect(1, "existing", "", "", 0)) Then

    rem Get an instance of the Component Interface;

    rem The Component Interface definition should already exist;

    &CI = &SESSION.GetComponent(Component.ABS_HIST);

If (&CI <> Null) Then

    rem Set required keys to GET the component;

    &CI.EMPLID = "8001";

    rem Instantiate the Component Interface;

    If (&CI.get()) Then

        rem Get a specific row in a collection;

        &ABS_HISTCollection = &CI.ABSENCE_HIST;

        &ABS_HISTItem = &ABS_HISTCollection.item(1);

        WinMessage(&ABS_HISTItem.BEGIN_DT | ", " |
&ABS_HISTItem.EMPLOYER_APPROVED);

        rem Set properties in the selected row;

        If (&ABS_HISTItem.BEGIN_DT = "09/04/1983") Then

            &ABS_HISTItem.BEGIN_DT = "09/03/1983";

            &ABS_HISTItem.EMPLOYER_APPROVED = "N";

        Else

            &ABS_HISTItem.BEGIN_DT = "09/04/1983";

            &ABS_HISTItem.EMPLOYER_APPROVED = "Y";

        End-If;

        rem Save changes to database;

```

```
        If (&CI.save()) Then

            WinMessage("Successfully Saved Component Interface.");

        Else

            WinMessage("Error occured in Save method.");

        End-If

    Else

        WinMessage("Error occured in Get method.");

    End-If;

Else

    WinMessage("Error occured in GetComponent.");

End-If;

Else

    WinMessage("Error occured in connect.");

End-If;
```

Java and Active Server Page Examples

The Java and Active Server Page (.asp) examples shown in this section use a component interface on the PTDMO demo database named BUS_EXP, which is based on the Business Expenses Component.

The following business expenses page allows the user to enter information about a type of expense, the amount, date, currency type, purpose, and so on.

Home > Administer Workforce > Administer Workforce (GBL) > Use > Business Expenses

Business Expenses

Bassani, Maria Miss Employee ID: 8901

Employee Business Expense Time View All First 1 of 1 Last

'Expense Period End Date: 07/25/2000 + -

Expense Period Total: USD

Business Expense Details View All First 1 of 1 Last

Charge Dt	Business Unit	'Expense Code Department	Expense Amount	'Currency	Business Purpose
07/25/2000	GBIBU	Mileage		USD	

Save Return to Search

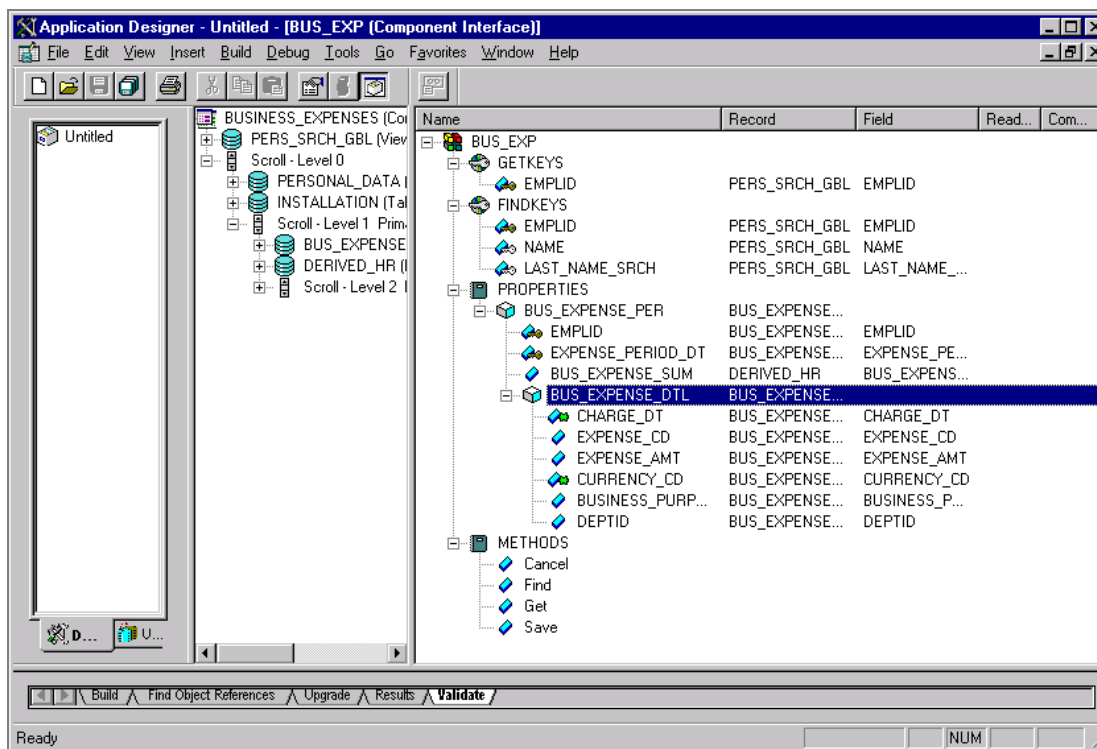
Business Expenses Page



Note. We recommend that before you create a component interface, you be familiar with the business rules, required fields, and acceptable input for a page. For example, when entering a new row in a Collection, a third party application needs to be configured so that the user enters all required fields before attempting to save or commit the row in PeopleSoft. If all required fields are not entered, the application will error when the user tries to invoke the Save() method.

The BUS_EXP component interface has two scroll levels just like the Business Expense Page. Because the NAME and LAST_NAME_SRCH fields are alternate search keys on the search record, just as on the page, they are not accessible in the same way as the properties. The Create() method is not available in the component interface because this page does not have the 'Add' mode enabled.

User-defined methods can only take simple types of arguments (such as number, character, and so on) because they are called from C/C++, COM, Visual Basic, Java, and PeopleCode. More complex types of arguments like rowset, array, and record are unknown to C/C++, COM and Visual Basic. All user-defined methods must return a value, even if it is only a dummy value.



Business Expense Component Interface Definition

Active Server Page Example

In this example, the page prompts the user to search for a record first. For example,

title test - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Real.com

Address http://eharris032000/webpub/test/ext_test/other/Opening_page.asp Go Links >>

Number of rows :4

Component Interface Demo: Business Expenses

Find Employee

<input checked="" type="checkbox"/>	Name	Starts With	Sch
<input type="checkbox"/>	Last Name	Starts With	
<input type="checkbox"/>	Employee ID	Starts With	

SEARCH

8001 - Schumacher, Simon (ASD) Submit

Done Local intranet

Business Expenses Start Page

This page comprises two HTML forms. When the user enters the search data and clicks Search, the *opening_page.asp* file populates the dropdown list. The following section shows the Visual Basic script with comments of the file.

Connecting to the Application Server

To access component interface, you need to establish a PeopleSoft session.

To create a session object, use the `server.CreateObject()` method. The `Connect` method, which takes five parameters, actually logs into a PeopleSoft session. Operator ID and password should not be hard-coded in the application, rather the user should be prompted at runtime. The `Connect()` method connects a session object to a PeopleSoft application server. If you already have a PeopleSoft session running, you must specify `EXISTING`, and not the `ConnectID:Port`. If you are using an existing connection to the application server, you cannot specify a different operator ID or password. If you do not specify these values as `NULL`, you must specify the exact same operator ID (and password) as the one that originally started the session.

Getting an Instance

Use the `GetComponent()` method with a session object to get an instance of a previously created component interface. Next, we want to search for an existing record by performing a search using primary or alternate search keys.

Code block 1 loads the search criteria from the previous call made by *opening_page.asp*.

```

<%

'code block 1

SEARCH_NAME = Request.Form("SEARCH_NAME")

NAME_SELECT = Request.Form("NAME_SELECT")

NAME_STRING = Request.Form("NAME_STRING")

SEARCH_LAST_NAME = Request.Form("SEARCH_LAST_NAME")

LAST_NAME_SELECT = Request.Form("LAST_NAME_SELECT")

LAST_NAME_STRING = Request.Form("LAST_NAME_STRING")

SEARCH_EMPLID = Request.Form("SEARCH_EMPLID")

EMPLID_SELECT = Request.Form("EMPLID_SELECT")

EMPLID_STRING = Request.Form("EMPLID_STRING")

```

Code block 2 makes a connection to the application server and gets the BUS_EXP component interface.

```

'code block 2

Set oSession = server.CreateObject ( "PeopleSoft.Session" )

nStatus = oSession.Connect(1, "//EHARRIS032000:9000", "PTDMO", "PTDMO", 0)

Set oBC = oSession.GetComponent("BUS_EXP")

```

Finding an Existing Record

After getting an instance of the component interface, we recommend you find what data instances you have access to using the Find() method. In a PeopleSoft search dialog, when a user enters an employee ID or name into the appropriate field and clicks the search button, the system performs the search. To accomplish this in a component interface, set the find keys for the component interface and then invoke the Find() method. This returns an object of type component interfaceCollection, which can be indexed to extract data such as the GetKeys.

Code block 3 sets the FindKeys for BUS_EXP.

```

'code block 3

oBC.EMPLID = EMPLID_SRCH_STRING

oBC.NAME = NAME_SRCH_STRING

oBC.LAST_NAME_SRCH = UCASE(LAST_NAME_SRCH_STRING)

```

Getting an Instance of Data

GetKeys are the key values required to return a unique instance of existing data. If the keys you specify allow for more than one instance of the data to be returned, or if no instance of the data matching the key values is found, there is a runtime error. Therefore, we recommend you use the Find() method to query the component interface for existing records prior to calling Get().

GetKeys can be set using simple assignment to the properties of the component interface and then the Get() method can be invoked. This will populate the component interface with data based on the key values you set; this is what has been referred to here as a data instance. The Get() method will return a Boolean value depending on its success or failure, however recovering this value for error handling is difficult in ASP because a failure of Get() causes an immediate runtime error in the script.

Code block 4 uses the find() method for the BUS_EXP object to return a collection of type BUS_EXPCollection.

```
`code block 4

SET BC_COLLECTION = oBC.find()

number_of_rows = BC_COLLECTION.COUNT

Response.Write("<BR> Number of rows :" & number_of_rows )
```

Code block 5 indexes through the collection that BUS_EXP.find() returned and uses the EMPLID and NAME to populate the dropdown list.

```
`code block 5

for counter_1 = 1 to number_of_rows

SET BC_TEMP = BC_COLLECTION.ITEM( Cint( counter_1))

emplid_temp = BC_TEMP.EMPLID

empl_name = BC_TEMP.NAME

lst_name_srch = BC_TEMP.LAST_NAME_SRCH

NEXT
```

Code block 6 creates a data instance of the component interface

```
`code block 6

oBC.EMPLID = emplid_temp

Status = oBC.Get()
```

»>

Migrating Through Scrolls

After getting a data instance, the next step will be to get access to the data in the component interface. PeopleSoft organizes data into scrolls so that a first-level scroll might have three rows of data in it and each of those rows may have several rows of data in them held in a second-level scroll. A user can examine data by moving the scroll bars up and down and looking at various data rows in the scrolls.

A scroll bar is similar to a *collection* in component interface, and rows of data in the *collection* are called *items*. The following screen shows how data is organized in the BUS_EXP component interface. Note the Properties of that component interface: there are two collections in the BUS_EXP component interface. The first one is BUS_EXPENSE_PER and the second one, nested below the first, is BUS_EXPENSE_DTL.

Name	Record	Field	Rea...	Com...
BUS_EXP				
GETKEYS				
EMPLID	PERS_SRCH_G...	EMPLID		
FINDKEYS				
EMPLID	PERS_SRCH_G...	EMPLID		
NAME	PERS_SRCH_G...	NAME		
LAST_NAME_SRCH	PERS_SRCH_G...	LAST_NAME...		
PROPERTIES				
BUS_EXPENSE_PER	BUS_EXPENSE...			
EMPLID	BUS_EXPENSE...	EMPLID		
EXPENSE_PERIOD...	BUS_EXPENSE...	EXPENSE_P...		
BUS_EXPENSE_S...	DERIVED_HR	BUS_EXPEN...		
BUS_EXPENSE_DTL	BUS_EXPENSE...			
CHARGE_DT	BUS_EXPENSE...	CHARGE_DT		
EXPENSE_CD	BUS_EXPENSE...	EXPENSE_CD		
EXPENSE_AMT	BUS_EXPENSE...	EXPENSE_A...		
CURRENCY_CD	BUS_EXPENSE...	CURRENCY_...		
BUSINESS_PU...	BUS_EXPENSE...	BUSINESS_P...		
DEPTID	BUS_EXPENSE...	DEPTID		
METHODS				
Cancel				
Find				
Get				
Save				

Structure of data in BUS_EXP Component Interface

It is possible to return a BUS_EXPENSE_PER collection through simple assignment as shown below. Then invoke the Count() method to determine how many Items are in the collection and returns one of those rows using the Item() method. Once we have an item out of a collection, data can be accessed in that item just as it would be in a page. This item could be used to change the EMPLID, EXPENSE_PERIOD, or BUS_EXPENSE_SUM of a row of data. Because there are two scroll levels, it is possible to repeat for a second scroll collection.

```
oBC.EMPLID = emplid_temp

Status = oBC.Get()

Set oBusExpPerCollection = oBC.BUS_EXPENSE_PER

Number_of_rows_in_collection_integer = oBusExpPerCollection.Count

Set BusExpPerItem =
    oBusExpPerCollection.Item(Cint(some_integer_variable))
```

```

Set BusExpenseDtlCollection = BusExpPerItem.BUS_EXPENSE_DTL

Number_of_rows_in_collection_integer = BusExpenseDtlCollection.Count

BusExpenseDtlItem = BusExpenseDtlCollection.Item(Cint( some_integer_variable ))

```

Editing and Accessing Data in an Item

Editing a data member of an Item can be accomplished using direct assignment or through the `setPropertyByName()` method, which returns a long. In either case, however, it is necessary to invoke the `Save()` method on the component interface object to commit changes.

Accessing data in an Item is nearly the same as editing it, and can also be accomplished in two ways. The first method is to use assignment as in the example below, and the second method is to use the `GetPropertyByName()` method. The `GetPropertyByName()` method usually returns a string even if that string represents a number.

```

detRow.EXPENSE_CD          = expense_cd

detRow.EXPENSE_AMT         = expense_amt

detRow.CURRENCY_CD         = currency_cd

i = detRow.setPropertyByName( "BUSINESS_PURPOSE", business_purpose )

i = detRow.setPropertyByName( "DEPTID", deptid)

oBC.Save()


expense_cd                  = detRow.EXPENSE_CD

expense_amt                 = detRow.EXPENSE_AMT

currency_cd                 = detRow.CURRENCY_CD

oRow.GetPropertyByName( "business_purpose" )

oRow.GetPropertyByName( "DEPTID" )

```

Inserting a Row into a Collection

```

Set detRow = detRows.InsertItem(Cint( some_integer_variable ))

```

Notice that the return value for inserting a new Item into a collection is the Item that was just inserted. After the row is inserted, edit all of the required fields in the item. If required fields are blank or data entered violates some business logic, the application will return a runtime error.

Deleting a Row from a Collection

Like inserting items, you can delete items using collection objects.

```
temp = oRows.DeleteItem(Cint(row_number))

oBC.Save()
```

The DeleteItem() method returns a boolean value according to the success or failure of the method and it is important to invoke the Save() method to commit the change to the database.

Disconnecting from a Session

After a session is no longer needed, disconnect from the application server. This is done by calling the disconnect() method on the session object.

```
call oSession.disconnect

set oSession = nothing
```

Java Example

Creating user interfaces without the limitations of HTML is a benefit to using Java. It is necessary for the classpath to be correctly set to include the libraries and the proper include statements must be made at the beginning of the class definition.

Hello World

File Edit Insert Help

Please Choose Search Options

Name ☒ Sch

Last Name ☐ Last Name

EmplID ☐ EmplID

Search Reset Submit

8001

Bus Purpose	Charge Date	Currency CD	Dept ID	Expense Amt	Expense Co...	Expense Dtl
Feedback	12/04/1998	USD	00001	15.69	03	USD
Not much	12/04/1998	USD	00001	3.69	03	USD
Air Fare	12/04/1998	USD	00001	5.69	03	USD
Something	12/04/1998	USD	00001	15.69	03	USD
Not much	12/04/1998	USD	00001	31.69	03	USD
inserted row	12/04/1998	USD	00001	15.25	03	USD
inserted row	12/04/1998	USD	00001	69.25	03	USD
Some row	12/04/1998	USD	00001	1411.63	03	USD
inserted row	12/04/1998	USD	00001	69.25	03	USD
Some row	12/04/1998	USD	00001	141.63	03	USD
Total:				1779.46		

12/12/1998

1

Insert Edit Delete

ComboBox2 Was changed to 1 end date is: 12/12/1998
 ComboBox2 Was changed to 8001 Skipping CI action
 ComboBox2 Was changed to 8001 Skipping CI action
 checkbox[0] Was changed

User Interface for Java Client

Connecting to the Application Server

Connecting to the application server in Java is similar to a connection with an .asp file. A C Adapter object is created and then used to create a C Session object. Also, the Connect() method is called as it was in .asp.

```
import PeopleSoft.ObjectAdapter.*;

import PeopleSoft.Generated.PeopleSoft.*;

import PeopleSoft.Generated.CompIntfc.*;

private ISession oSession;

private CAdapter oAdapter;

oAdapter = new CAdapter();

oSession = new CSession(oAdapter.getSession());

oSession.Connect(1,"//EHARRIS032000:9000","PTDMO","PTDMO",new byte[0]);
```

Getting an Instance of the Component Interface

Getting an instance of a component interface is almost identical in Java and .asp. All of the same rules apply, and the component interface definition must exist or the application will error.

```
busExpense = new CBusExp( oSession.GetComponent( "BUS_EXP" ) );
```

Finding an Existing Record

You can query a component interface to find what data instances are possible based on primary and alternate search keys.

```
busExpense.setName( searchDialogStrings[ 0 ] );

busExpense.setLastNameSrch( searchDialogStrings[ 1 ] );

busExpense.setEmplid( searchDialogStrings[ 2 ] );

return( busExpense.Find() );
```

Although it looks different, the code above does the exact same things in Java as code lines did in .asp.

Getting an Instance of Data

```
busExpense.setEmplid( getKey );

boolean result = busExpense.Get();
```

Migrating Through Scrolls

The following code lines set up the connection to the application server and get the component interface.

```
oAdapter = new CAdapter();

oSession = new CSession(oAdapter.getSession());

oSession.Connect(1,"/EHARRIS032000:9000","PTDMO","PTDMO",new byte[0]);

busExpense = new CBusExp( oSession.GetComponent( "BUS_EXP" ) );

busExpense.setEmplid( getKey );

boolean result = busExpense.Get();

busExpenseFirstScrollItemCollection = busExpense.getBusExpensePer();

busExpenseFirstScrollItem = firstScrollCollection.Item( firstScrollIndex );

return( busExpenseFirstScrollItem.getBusExpenseDtl() );
```

Editing and Accessing Data in an Item

Editing and accessing data in Java is very similar to what is done in .asp, however, accessing private data members of a given object is not an option due to encapsulation of well written Java classes. Therefore, Java code will rely on the public members of the class rather than direct assignment. However, there is still more than one way access data in an Item.

```
long j = busExpenseSecondScrollCollection.getCount();

Object [][] data = new Object[ ((int)j + 1) ][ 7 ];

for( int i = 1; i < j + 1 ; i++ )

{

busExpenseSecondScrollItem = busExpenseSecondScrollCollection.Item( i );

data[(i - 1)][0] = busExpenseSecondScrollItem.getBusinessPurpose();

data[(i - 1)][1] = busExpenseSecondScrollItem.getChargeDt();

data[(i - 1)][2] = busExpenseSecondScrollItem.getCurrencyCd();

data[(i - 1)][3] = busExpenseSecondScrollItem.getDeptid();

data[(i - 1)][4] = busExpenseSecondScrollItem.getExpenseAmt();
```

```

data[(i - 1)][5] =
busExpenseSecondScrollItem.GetPropertyByName("ExpenseCd");

data[(i - 1)][6] = busExpenseSecondScrollItem.GetPropertyByName("CurrencyCd");

}return( data );

```

In the following example, data is accessed using the `getName_OF_PROPERTY()` method of an Item or by using the generic `getPropertyByName()` method. These code lines show how an entire collection of data can be captured and packaged into an Object for transfer to a calling Object.

```

busExpenseFirstScrollItem.setEmplid( emplid );

busExpenseFirstScrollItem.setExpensePeriodDt( expensePeriodDt );

return( busExpense.Save() );

```

Just as before, data is edited using Item objects and using the `setNameOfProperty()` method of those Items. Also, note that we needed to call the `Save()` method on the component interface to commit the changes.

Inserting an Item into a Collection

```

busExpenseSecondScrollItem = busExpenseSecondScrollCollection.InsertItem(
secondScrollIndex );

```

Collection objects in Java also have the `InsertItem()` method where the return value is the Item just inserted. After a new Item is created, simply edit data in it and then remember to call the `Save()` method to commit the changes.

Deleting a Row from a Collection

```

busExpenseSecondScrollCollection.DeleteItem( secondScrollIndex );

boolean result = busExpense.Save();

```

Remember to save after the delete method is called to commit changes.

Disconnecting from a Session

After a session is no longer needed, it should disconnect from the application server. This is done by calling the `disconnect()` method on the session object.

```

oSession.Disconnect();

```


CHAPTER 5

Component Interface SDK

This section lists the steps that create a program calling a component interface to communicate synchronously with a PeopleSoft application. The PeopleSoft Integration Software Development Kit (SDK) installed with your application includes a sample project with data and source code you can use to test your development work.



For more information and an overview of the PeopleSoft Integration SDK, see Integration Software Development Kit.

Requirements

You will need the following to call a PeopleSoft component interface.

- Working understanding of C++, COM or Java.
- Specific description of the component interface being called. If PeopleSoft provides your component interface, look in the “EIP Catalog” section of the *Enterprise Integration PeopleBook* for the description; otherwise, contact the custom component interface developer.
- Connection to the PeopleSoft Object Adapter that is installed on a PeopleSoft Application Server.

The PTSDK Development Project

PeopleSoft uses the Business Expense component as a working example for the PTSDK development project. You can import this component into any PeopleSoft database to run all of the samples provided. The following table lists the files in the PTSDK project and installed in the PeopleSoft home directory (PS_HOME), under **sdk\sdkdb**.

Name	Description
PTSDK	Folder containing all the files required to import the PTSDK project to Application Designer
PTSDKData.dat	Sample data to populate SDK_BUS_EXP tables
PTSDKDataImport.dms	Data Mover Script to import sdk_data.dat into PeopleSoft



The PTSDK project and associated data are for development purposes only and will not be supported by PeopleSoft.

PTSDK Project Objects

Object Type	Name	Description
Component	SDK_BUS_EXPENSES	Component for SDK
Component interface	SDK_BUS_EXP	Component Interface for component
Message definition	SDK_BUS_EXP_APPR_MSG	Application Message for component
Message channel	SDK_BUS_EXP_MSG_CHANNEL	Application Message Channel
Page	SDK_BUS_EXPENSES	HTML page for component
Page	SDK_PERS_SRCH_SBP	HTML search page

PTSDK Records

Record	Field	Description
SDK_BUS_EXP_DTL	SDK_EMPLID	Employee ID
	SDK_EXP_PER_DT	Expense Period Date
	SDK_CHARGE_DT	Charge Date
	SDK_EXPENSE_CD	Expense Code
	SDK_EXPENSE_AMT	Amount of expense
	SDK_CURRENCY_CD	Currency of expense
	SDK_BUS_PURPOSE	Purpose of expense
	SDK_DEPTID	Department ID

Record	Field	Description
SDK_BUS_EXP_PER	SDK_EMPLID	Employee ID
	SDK_EXP_PER_DT	Expense Period Date
	SDK_SUBMIT_FLG	Submitted Flag
	SDK_INTL_FLG	
	SDK_APPR_STATUS	Approval Status
	SDK_APPR_INSTANCE	Approval Instance
	SDK_DESCR	Description
	SDK_COMMENTS	Comments
SDK_COMPANY_TBL	SDK_COMPANY	Company Name
	SDK_EFFDT	Effective Date
	SDK_EFF_STATUS	Effective Date Status
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
SDK_COUNTRY_TBL	SDK_COUNTRY	Country
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
SDK_CUR_RT_TYPE	SDK_CUR_RT_TYPE	Currency Rate Type
	SDK_EFFDT	Effective Date
	SDK_EFF_STATUS	Effective Date Status
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
SDK_CURR_CD_TBL	SDK_CURRENCY_CD	Currency Code
	SDK_EFFDT	Effective Date
	SDK_EFF_STATUS	Effective Date Status
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
SDK_DEPT_TBL_SBR	SDK_EEO4_FUNCTION	

Record	Field	Description
SDK_DEPT_TBL	SDK_DEPTID	Department ID
	SDK_EFFDT	Effective Date
	SDK_EFF_STATUS	Effective Date Status
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
	SDK_COMPANY	Company Name
	SDK_LOCATION	Location
	SDK_MANAGER_ID	Manager ID
	SDK_MANAGER_POSN	Manager Position
	SDK_BUDGET_LVL	Budget Level
	SDK_DEPT_TBL_SBR	Department Table Subrecord
SDK_DEPT_TBL_VW	SDK_DEPTID	Department ID
	SDK_EFFDT	Effective Date
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
SDK_DERIVED	SDK_EMPLID	Employee ID
	SDK_EMPL_RCD	Employee Record
	SDK_EMPLID_OLD	
	SDK_EMPLID_PROCESS	
	SDK_BUS_EXP_SUM	Business Expense Sum
	SDK_EFFDT	Effective Date
	SDK_CAR_MODEL_DESC	Car Model Description
SDK_INSTALL	SDK_POSITION_MGMT	Position Management
	SDK_COUNTRY	Country
	SDK_EXCHNG_TO_CURR	Exchange to Currency
	SDK_EXCHNG_RT_TYPE	Exchange to Rate Type
	SDK_GER	
SDK_INTL_FLG_CD	SDK_INTL_FLG	Effective Date
	SDK_EFFDT	Effective Date Status
	SDK_EFF_STATUS	Description
	SDK_DESCR	Short Description
	SDK_DESCRSHORT	

Record	Field	Description
SDK_JOB	SDK_EMPLID	Employee ID
	SDK_EMPL_RCD	Employee Record
	SDK_EFFDT	Effective Date
	SDK_EFFSEQ	Effective Sequence
	SDK_DEPTID	Department ID
SDK_LOCH_TBL	SDK_LOCATION	Location
	SDK_EFFDT	Effective Date
	SDK_EFF_STATUS	Effective Date Status
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
SDK_PER_SGBLSBR	SDK_EMPLID	Employee ID
	SDK_OPERCLASS	Operator Class
	SDK_EMPL_RCD	Employee Record
	SDK_NAME	Employee Name
	SDK_LAST_NAME_SRCH	Last Name Search
	SDK_ACCESS_CD	Access Code
SDK_PER_SRCHGBL	SDK_PER_SGBLSBR	Social Security Number
	SDK_SSN	
	SDK_SIN	
	SDK_NAT_INS_CD	
	SDK_SSN_FRA	
	SDK_SIN_GER	
	SDK_NATIONAL_ID	

Record	Field	Description
SDK_PERS_DATA	SDK_EMPLID	Employee ID
	SDK_NAME	Employee Name
	SDK_LAST_NAME_SRCH	Last Name Search
	SDK_FIRST_NAME_SRC	First Name Search
	SDK_PER_STATUS	
	SDK_ORIG_HIRE_DT	Original Hire Date
	SDK_SEX	Employees Sex
	SDK_BIRTHDATE	Employees Birth Date
	SDK_BAS_DATA_CHG	
	SDK_PER_TYPE	
	SDK_NATIONAL_ID	
	SDK_SSN	
	SDK_SIN_GER	Social Security Number
	SDK_SIN	
	SDK_SSN_FRA	
	SDK_NAT_INS_CD	
SDK_POS_DATA	SDK_POSITION_NBR	Position Number
	SDK_EFFDT	Effective Date
	SDK_EFF_STATUS	Effective Date Status
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description
	SDK_DEPTID	Department ID
SDK_PSTREENODE	SDK_SETID	Setid
	SDK_TREE_NAME	Tree Name
	SDK_EFFDT	Effective Date
	SDK_TREE_NODE_NUM	Tree Node Number
	SDK_TREE_NODE	Tree Node
	SDK_TREE_NODE_END	Tree Node Number End
SDK_RT_TYPE_TBL	SDK_RT_TYPE	Rate Type
	SDK_DESCR	Description
	SDK_DESCRSHORT	Short Description

Record	Field	Description
SDK_SCRTY_DEPT	SDK_OPRID	Operator ID
	SDK_DEPTID	Department ID
	SDK_ACCESS_CD	Access Code
	SDK_TREE_EFFDT	Effective Date
	SDK_TREE_NODE_NUM	Tree Node Number
	SDK_TREE_NODE_END	Tree Node Number End

SDK_BUS_EXPENSES Test Page

Use the SDK test page called SDK_BUS_EXPENSES to test whether the component interface (SDK_BUS_EXP) updated the database correctly.

Home > Administer Workforce > Administer Workforce (GEL) > Use > Sdk Bus Expenses

Sdk Bus Expenses

THIS PAGE IS PART OF THE iSDK AND IS NOT FOR BUSINESS USE

Schumacher, Simon ID: 8001

View All First 1 of 1 Last

Expense Period End Date: 07/14/2000

Expense Period Total: 632.00 USD

Charge Dt	Expense Code	Expense Amount	Business Purpose	Department
07/14/2000	Meals	34.00 USD	Corporate Meeting	10400
07/18/2000	Car Rental	465.00 USD	Corporate Meeting	00001
01/01/2001	Refer Fee	123.00 USD	Corporate Meeting	00001

Save Return to Search

SDK_BUS_EXPENSES page

Installing the PTSDK Project

The following steps describe how to install the objects and create the SDK development database.

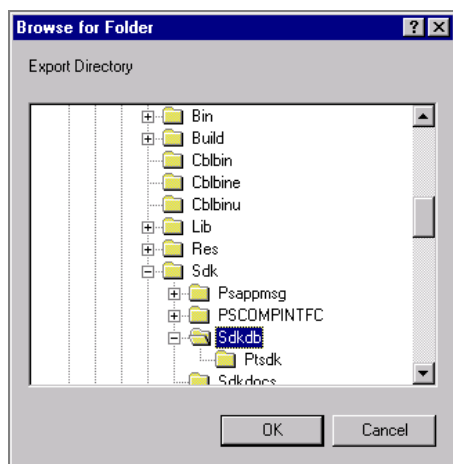
To install the sample SDK project:

1. Open Application Designer.
2. Select File, Copy Project From File.



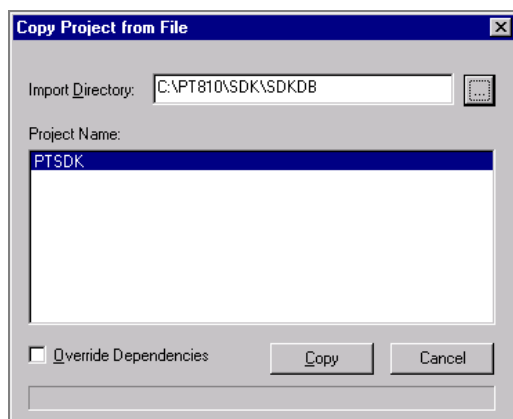
For more information about using Copy Project From File, see Application Designer.

3. Browse to the <PS_HOME>\sdk\sdkdb folder.



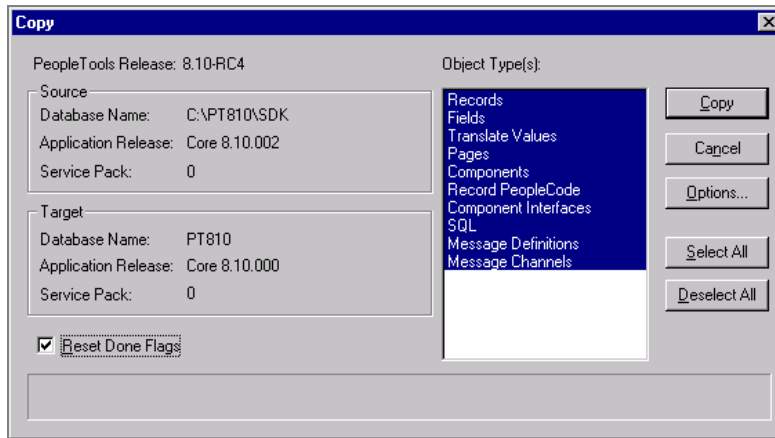
Finding the Sdkdb Folder

4. Select the folder and click **OK**.



Copy PTSDK Project from File

5. Select the Project PTSDK that contains all the SDK objects and click **Copy**.



Copy Source to Target Database

6. Click **Copy** to copy the objects into your database.

The PTSDK project appears in the project window.

To create the database tables:

7. Select **Build, Project**.

The Build dialog box will appear.

8. Select the **Create Tables** and **Create Views** checkboxes.
9. Select **Execute SQL Now** in the Build Execute Options and click the Build button.

This builds the database tables and views for the SDK.

10. Add the SDK_BUS_EXPENSES component to a menu.

You must do this before the page can be viewed.

11. Give Security Permission to the component and component interface.

Use the Administer Security page to allow access to the SDK_BUS_EXPENSES component and to the SDK_BUS_EXP component interface.



For more information about setting security permissions, see Security.

12. Run the Data Mover script **import_script.dms** on **sdk_data.dat**.

Both files can be found in <PS_HOME>\sdk\sdkdb. This step imports sample data into the database for use during development.



For more information about using the Data Mover, see Data Mover.

Component Interface Tester and Samples

The SDK includes a component interface, called SDK_BUS_EXP, which is part of the sample development project delivered with the SDK. You can use it as follows:

- The Component Interface (CI) Tester is a simple utility you can use to test your external connection to SDK_BUS_EXP through the PeopleSoft application server.
- The external integration samples also test your connection, but additionally enable you to test synchronous data entry and retrieval to and from the SDK database using SDK_BUS_EXP.

The CI Tester and the samples are provided as source code. They're available in four different languages—C++, Visual Basic, ASP, and Java.



The source files mentioned in this section are located relative to the installed PeopleSoft home directory (PS_HOME). You must install the sample development project to use the tester and the samples.



For more information about installing PTSDK, the sample development project, see The PTSDK Development Project .

C++ Tester and Sample

The C++ files include project files, project workspaces, source code and header files. Comments are listed in the code to explain each function. The file locations listed below are relative to <PS_HOME>\sdk\pscompintfc\src\C++\samples.

Filename	Location	Type
pscitester.dsp	pscitester	Project file
pscitester.dsw	pscitester	Project workspace
pscitester.cpp	pscitester	Source file
sdk_bus_exp.dsp	sdk_bus_exp	Project file
sdk_bus_exp.dsw	sdk_bus_exp	Project workspace
sdk_bus_exp.cpp	sdk_bus_exp	Source file
StdAfx.cpp	inc	Source file
apiadapterdef.h	inc	Header file
cidef.h	inc	Header file

Filename	Location	Type
peoplesoft_peoplesoft_i.h	inc	Header file
StdAfx.h	inc	Header file

Preparing Your C++ Tester and Sample

To prepare your workstation:

1. Install the external API, EXTAPI.

Refer to the *PeopleSoft 8 Installation and Administration Guide*, Chapter 11, External Integration Installations.

2. Set the client path environment variable to point to **psapiadapter.dll** in EXTAPI.

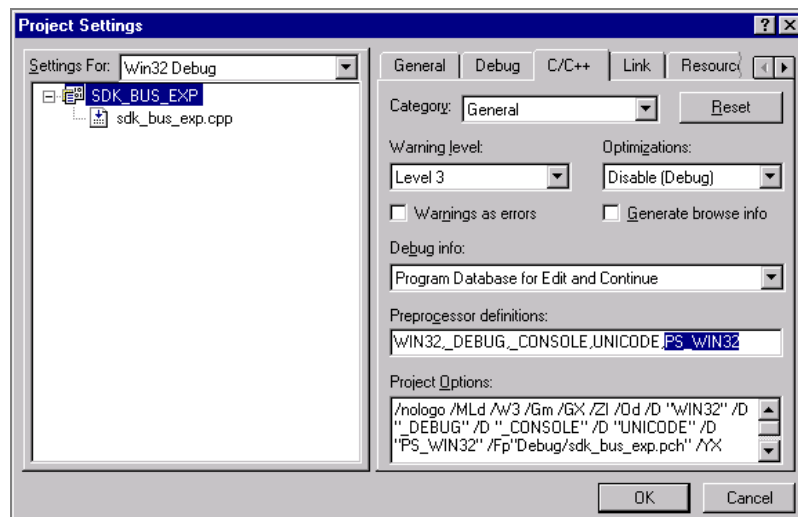
To configure your compiler for the C++ project:

3. In Visual C++, open the Project Settings dialog box and select the C/C++ tab.



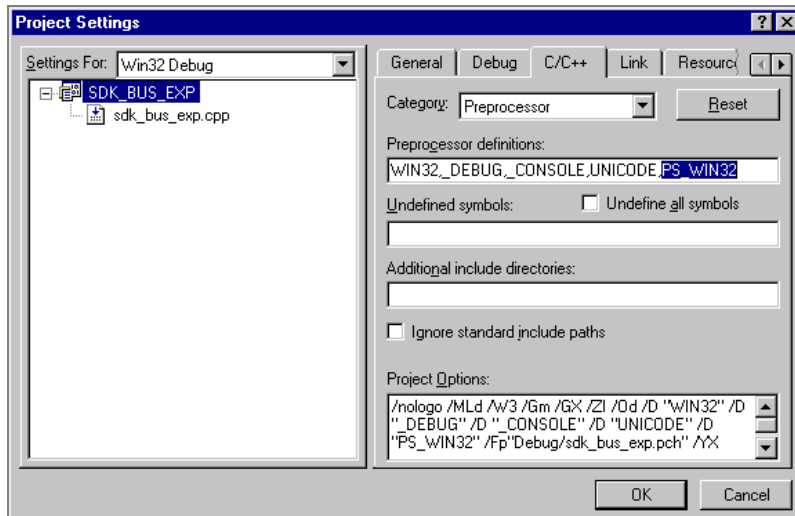
These instructions assume you're using Microsoft Visual C++. If you use a different compiler, apply the equivalent settings for that product.

4. Select the General Category and add PS_WIN32 to the Preprocessor definitions.



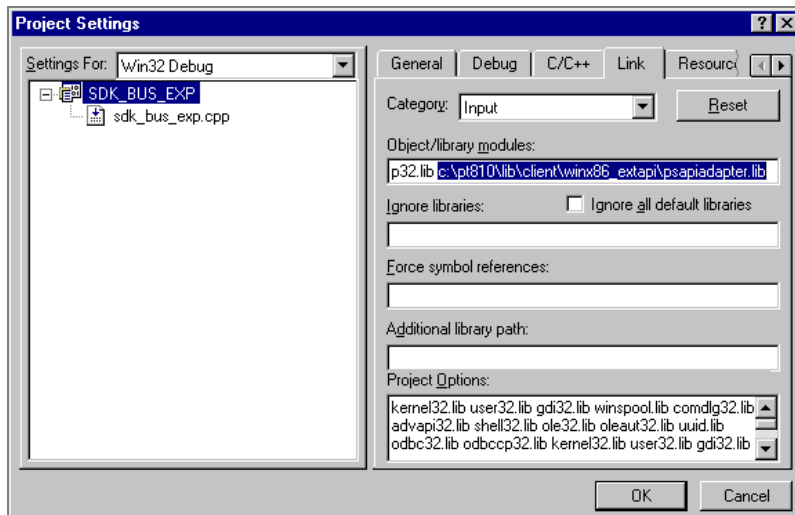
Set Preprocessor definitions for the C/C++ General Category

5. Select the Preprocessor Category and add PS_WIN32 to the Preprocessor definitions.



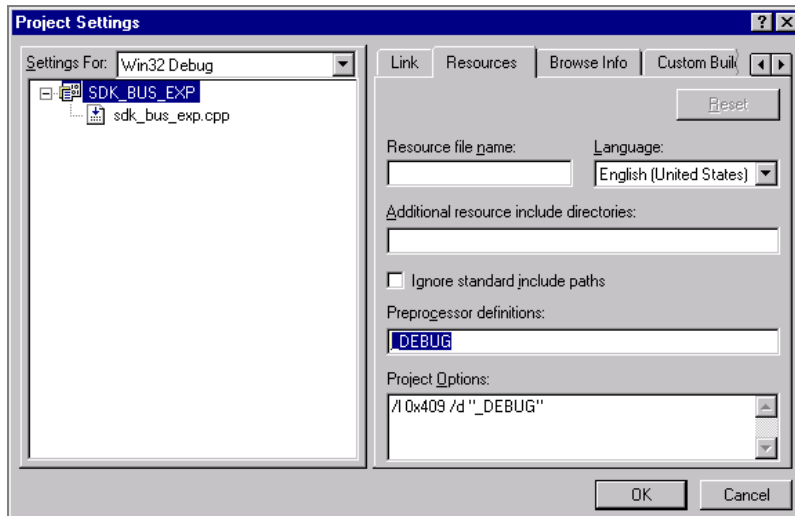
Set Preprocessor definitions for the C/C++ Preprocessor Category

6. Go to the Link tab, and select the Input Category.



Set Object/library modules path (the path shown is example only)

7. Specify the full path to **psapiadapter.lib** to the Object/library modules.
8. Go to the Resources tab and add `_DEBUG` to the Preprocessor definitions.



Set Preprocessor definitions on the Resources tab

Using the C++ CI Tester

The CI tester is run from the command line.

To run the compiled C++ CI Tester:

1. In a DOS window, change directories to the location of the C++ CI tester directory, **<PS_HOME>\sdk\pscompintfc\src\C++\samples\pscitester**.
2. Enter **pscitester** on the command line.

You'll be prompted for parameters one at a time.

3. At each prompt, type the appropriate value and press **Enter**:

Enter The Application Server Name [localmachinename]:

Enter The Application Server Port Number [9000]:

Enter PeopleSoft UserID [PTDMO]:

Enter PeopleSoft UserID Password [PTDMO]:

If the connection is successfully established, you'll see the message **"Connected to Appserver. . ."**, followed by a system prompt.

```

C:\WINNT\System32\cmd.exe
C:\JOA\C\psctestester\Debug>psctestester
Application Server Connect Information...
Enter The Application Server Name [BOINEZA101999]: BOINEZA101999
Enter The Application Server Port Number [9000]: 9000
Enter PeopleSoft UserID [PTDMO]: PTDMO
Enter PeopleSoft Password [PTDMO]:
Connected to Appserver...
C:\JOA\C\psctestester\Debug>

```

CI tester — C++ version

Using the C++ CI Sample

This sample is run from the command line.

To run the compiled C++ sample:

1. In a DOS window, change directories to the location of the C++ sample directory, **<PS_HOME>\sdk\pscompintfc\src\C++\samples\sdk_bus_exp**.
2. Enter **sdksdk_bus_exp** on the command line.

You'll be prompted for parameters one at a time.

3. At each prompt, type the appropriate value and press Enter:

Enter Server Name [//localmachinename]: (Application Server name)

Enter Port Number [9000]: (Application Server JSL port number)

Enter PeopleSoft User ID [PTDMO]:

Enter PeopleSoft Password [PTDMO]:

You must provide least one of the following three search parameters. If you just press Enter for all three, the program will exit. Incremental searches are available (e.g. ID=8 will return all ID's starting with 8):

Enter Employee ID:

Enter Employee Name <optional>:

Enter Employee Last Name <optional>:

The list of employees produced by the search will appear.

4. Enter the **Employee ID** of an employee on the list.

The business expense details for the selected employee will be displayed.

```

C:\WINNT\System32\cmd.exe
C:\JOA\C\sdk_bus_exp\Debug>sdk_bus_exp

Application Server Connect Information...
Enter The Application Server Name [ROINEZA101999]:
Enter The Application Server Port Number [9000]:
Enter PeopleSoft UserID [PTDM01]:
Enter PeopleSoft UserID Password [PTDM01]:

Search for Employees...
Enter Employee ID: 80
Enter Employee Name (optional):
Enter Employee Last Name (optional):

Employee Number      Name      Last Name
=====
      8001      Schumacher, Simon      ASD
      8052      Avery, Joan      AVERY

Select Employee ID from this list.
Enter Employee ID: 8001

Expense Period Number: 1
Expense Period End Date: 11/10/1999
Expense Period Total: 0.00

Charge Dt. Expense Code Expense Amount      Business Purpose      Department
=====
11/13/2000      0.00

Update Expense Period Row (y/n) [y]:N

```

CI sample — C++ version

Visual Basic Tester and Sample

This VB sample has a simple GUI to allow for data entry and retrieval. The file locations listed below are relative to <PS_HOME>\sdk\pscompintfc\src\com\samples\vb:

Filename	Location	Type
pscitester.frm	pscitester	Form definition
pscitester.vbp	pscitester	Visual project file
SDK_BUS_EXP.frm	sdk_bus_exp	Form definition
SDK_BUS_EXP.vbp	sdk_bus_exp	Visual project file

Preparing Your Visual Basic Tester and Sample

To prepare your workstation:

1. Install the external API, EXTAPI.

Refer to the *PeopleSoft 8 Installation and Administration Guide*, Chapter 11, External Integration Installations.

2. Set the client path environment variable to point to **psapiadapter.dll** in EXTAPI.

Using the Visual Basic CI Tester

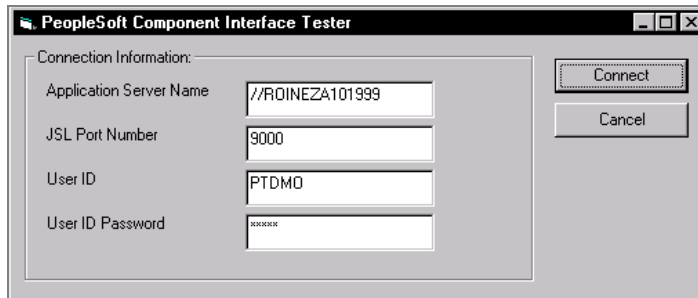
To run the compiled Visual Basic CI tester:

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

<PS_HOME>\sdk\pscompintfc\src\com\samples\VB\pscitester.

2. Enter **pscitester** on the command line.

The initial form will appear.



CI tester — Visual Basic version

3. Enter the Application Server Name.
4. Enter the application server **JSL Port Number** (9000).
5. Enter the PeopleSoft **User ID** (PTDMO).
6. Enter the PeopleSoft **User ID Password** (PTDMO).
7. Click **Connect** to test the connection.

If the connection is successfully established, you'll see the message "**Connection to the Application Server succeeded**".



Visual Basic confirmation message

Using the Visual Basic CI Sample

To run the compiled Visual basic sample:

1. In a DOS window, change directories to the location of the VB sample directory, <PS_HOME>\sdk\pscompintfc\src\com\samples\VB\sdk_bus_exp.
2. Enter **sdk_bus_exp** on the command line.

The initial form will appear.

The screenshot shows the 'SDK Business Expense' window. It has a title bar with standard window controls. The main area is divided into three sections: 'Connection Information', 'Search Keys', and 'Search Results'. The 'Connection Information' section contains four input fields: 'Application Server:' (with text '//ROINEZA101999'), 'Port No.:' (with text '9000'), 'User ID:' (with text 'PTDMO'), and 'Password:' (with masked text '*****'). The 'Search Keys' section contains three input fields: 'Employee ID:' (with text '80'), 'Name:', and 'Last Name:'. To the right of these fields are 'Search' and 'Cancel' buttons. The 'Search Results' section contains a table with three columns: 'Employee', 'Name', and 'Last Name'. The table is currently empty.

Initial form for the SDK_BUS_EXP Visual Basic sample

3. Enter the Application Server name.
4. Enter the application server JSL **Port Number** (9000).
5. Enter the PeopleSoft **User ID** (PTDMO).
6. Enter the PeopleSoft **Password** (PTDMO).
7. Enter at least one of the following: the **Employee ID**, **Name** or **Last Name**.
8. Click **Search**.

The list of employees produced by the search will appear.

The screenshot shows the 'SDK Business Expense' window after a search. The 'Search Keys' section is the same as in the previous image. The 'Search Results' section now displays a table with two rows of data:

Employee	Name	Last Name
8001	Schumacher, Simon	ASD
8052	Avery, Joan	AVERY

Employee search results in the Visual Basic sample

9. Double-click an employee name.

The business expense details for that employee will appear.

SDK Business Expense

Connection Information:

Application Server: //R0INEZA101999 Port No.: 9000 User ID: PTDMO Password: *****

Search Keys:

Employee ID: 80 Name: Last Name:

Search Results:

Employee	Name	Last Name
8001	Schumacher, Simon	ASD
8052	Avery, Joan	AVERY

Business Expense Details for Employee ID: 8001

Expense Period End	Expense Period Total	Charge Date	Expense Code	Expense Amount	Business Purpose	Department
10/11/2000	955					
		01/01/2000	01	300	Conference	00001
		01/05/2000	01	555	Conference	00001

Insert Business Expense Details:

Business Expense Period: 10/11/2000

Charge Date: Expense Code: Expense Amount: Business Purpose: Department ID:

Save Exit

Business expense details for the selected employee

ASP Tester and Sample

The ASP files consists of 6 separate ASP pages. The file paths listed below are relative to <PS_HOME>\sdk\pscompintfc\src\com\samples\asp:

<i>File Path and Name</i>	<i>Description</i>
psctest\psctest.asp	This CI tester form accepts connection parameters for testing.
sdk_bus_exp\SDK_BUS_EXP.asp	The entry page to sign on to the SDK Business Expense sample. Upon providing the Application Server Connect information and the key field values, a listing of Employee IDs is created.
sdk_bus_exp\SDK_BUS_EXP_SEARCH_LIST.asp	This page lists the Employee IDs for the key values provided in SDK_BUS_EXP.asp
sdk_bus_exp\SDK_BUS_EXP_ADD_DETAILS.asp	This page accepts the SDK Business Expense Details field values and sends the data to SDK_BUS_EXP_SAVE_DETAILS.asp to

File Path and Name	Description
	be saved.
sdk_bus_exp\SDK_BUS_EXP_LIST_DETA ILS.asp	This page to lists all the SDK Business Expense Periods and details for the selected Employee ID.
sdk_bus_exp\SDK_BUS_EXP_SAVE_DET AILS.asp	This page performs the insert of the SDK Business Expense Details line and saves the data.
sdk_bus_exp\SDK_BUS_EXP_FUNCLIB.as p	This file contains generic functions that are used by the other ASP pages.

Preparing Your ASP Tester and Sample

To prepare your workstation:

1. Install the external API, EXTAPI.

Refer to the *PeopleSoft 8 Installation and Administration Guide*, Chapter 11, External Integration Installations.

2. Set the client path environment variable to point to <PS_HOME>\bin\client\winx86_extapi.
3. Install Microsoft IIS.

Using the ASP CI Tester

To run the ASP CI tester:

1. Open pscitester.asp in a Web browser.

The initial form will appear.

CI tester — ASP version

2. Enter the Application Server Name.
3. Enter the Application Server Jolt Port number (9000).
4. Enter the **PeopleSoft User ID** (PTDMO).
5. Enter the **PeopleSoft User ID Password** (PTDMO).
6. Click **Submit**.

If the connection is successfully established, you'll see the message "**Connect to session passed**".



ASP confirmation message

Using the ASP CI Sample

To run the ASP sample:

1. Open SDK_BUS_EXP.asp in a Web browser.

The initial form will appear.

SDK Business Expenses Component Interface - Search

Connection Information:

Application Server Name	Application Server Jolt Port	PeopleSoft User ID	User ID Password
//roineza101999	9000	PTDMO	PTDMO

Search Keys:

Employee ID

Name

Last Name

Initial form for the SDK_BUS_EXP ASP sample

2. Enter the Application Server Name.
3. Enter the Application Server Jolt Port number (9000).
4. Enter the **PeopleSoft User ID** (PTDMO).
5. Enter the **PeopleSoft User ID Password** (PTDMO).
6. Enter at least one of the following: the **Employee ID**, **Name** or **Last Name**.
7. Click **OK**.

The search results matching your entered keys will appear.

SDK Business Expenses Component Interface - List

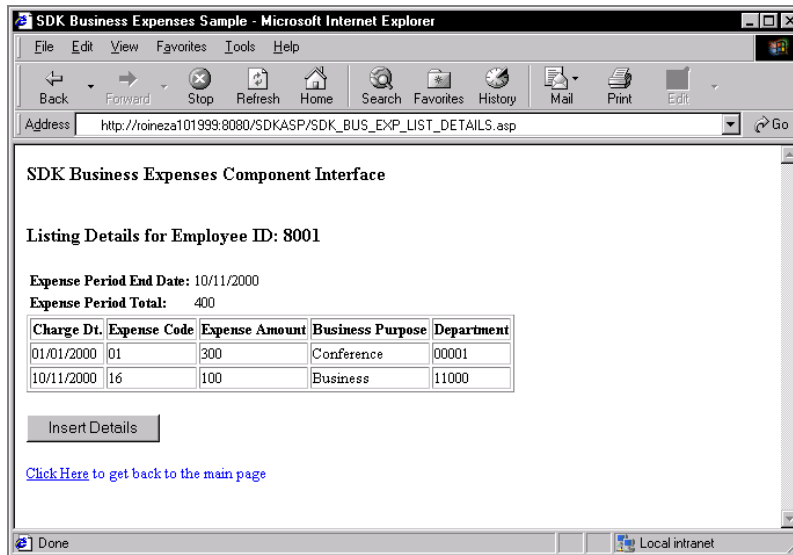
Employee ID	Name	Last Name	Select
8001	Schumacher, Simon	ASD	<input type="button" value="Select"/>
8052	Avery, Joan	AVERY	<input type="button" value="Select"/>

[Click Here to get back to the main page](#)

Employee search results in the ASP sample

8. Click **Select** next to an employee name.

The business expense details for that employee will appear.



Business expense details for the selected employee

Java Tester and Sample

The Java source code is in two files:

- <PS_HOME>\sdk\pscompintfc\src\java\samples\pscitester\pscitester.java
- <PS_HOME>\sdk\pscompintfc\src\java\samples\sdk_bus_exp\sdk_bus_exp.java

For Java bindings EXTAPI is not required; instead use the Java Object Adapter (JOA) shipped with your PeopleSoft application.

Preparing Your Java Tester and Sample

To prepare your workstation:

1. On the Environment tab of the System control panel, add the following path to the CLASSPATH environment variable:

<PS_HOME>\web\psjoa\psjoa.jar

2. Install the Sun JVM.

Using the Java CI Tester

The CI tester is a command line program.

To run the compiled Java CI Tester:

1. In a DOS window, change directories to the location of the Java CI tester directory, **<PS_HOME>\sdk\pscompintfc\src\java\samples\pscitester**.
2. Launch the executable with:

```
java pscitester.pscitester
```

You'll be prompted for parameters one at a time.

3. At each prompt, type the appropriate value and press **Enter**:

Enter The Application Server Name:

Enter The Application Server Port Number: (9000)

Enter PeopleSoft UserID: (PTDMO)

Enter PeopleSoft UserID Password: (PTDMO)

If a connection is successfully established, you'll see a message confirming the connection.

Using the Java CI Sample

This sample is a command line program.

To run the compiled Java CI Tester:

1. In a DOS window, change directories to the location of the Java sdk_bus_exp directory, **<PS_HOME>\sdk\pscompintfc\src\java\samples\pscitester**.
2. Launch the executable with:

```
java sdk_bus_exp.sdk_bus_exp
```

You'll be prompted for parameters one at a time.

3. At each prompt, type the appropriate value and press **Enter**:

Enter The Application Server Name:

Enter The Application Server Port Number: (9000)

Enter PeopleSoft UserID: (PTDMO)

Enter PeopleSoft UserID Password: (PTDMO)

You must provide least one of the following three search parameters. If you press Enter for all three, the program will exit. Incremental searches are available (e.g. ID=8 will return all ID's starting with 8):

Enter Employee ID:

Enter Employee Name (optional):

Enter Employee Last Name (optional):

4. Enter the **Employee ID** of an employee on the list.

The business expense details for the selected employee will be displayed.

```

C:\WINNT\system32\CMD.EXE - java sdk_bus_exp.sdk_bus_exp

Application Server Connect Information...
Enter The Application Server Name:
ROINEZA101999
Enter The Application Server Port Number:
9000
Enter PeopleSoft UserID:
PTDMO
Enter PeopleSoft UserID Password:
PTDMO

Search for Employees...
Enter Employee ID:
80
Enter Employee Name <optional>:
Enter Employee Last Name <optional>:

Employee ID      Employee Name
-----
8001             Schumacher, Simon
8052             Avery, Joan

Select Employee ID from this list...
Enter Employee ID:

```

CI sample — Java version

Index

A

- access to component interface 2-10
 - from COM programs 3-3
- active server page
 - in component interface 4-6
- API for component interface 1-2
- architecture
 - component interface 1-2
- attributes of
 - component interface 1-3

C

- CD-ROM
 - ordering ii
- collections
 - component interface 1-4
- COM library area
 - setting options 3-1
- component interface
 - access from COM programs 3-3
 - access from PeopleCode 4-2
 - access to 2-10
 - active server page example 4-6
 - adding keys 2-7
 - and components 1-2
 - API 1-2
 - architecture 1-2
 - attributes 1-3
 - building PeopleSoft APIs 3-1
 - calling another component interface 3-6
 - component interface view 2-2
 - ComponentName property 1-5
 - connecting to 3-3
 - CopyRowsetDelta method 1-7
 - create method 1-6
 - CreateKeyInfoCollection property 1-5
 - creating 2-3
 - creating properties 2-5
 - CurrentItem method 1-8
 - CurrentItemNum method 1-8
 - data collection methods 1-8
 - DataRow methods 1-8
 - definition 1-1
 - DeleteItem method 1-7
 - differences from online behavior 3-5
 - email 3-6
 - example of creating one 4-1

- example of VB template file 2-21
- extapi directory 3-3
- FindKeyInfoCollection property 1-5
- generating PeopleCode 2-24
- generating VB template 2-21
- GetEffectiveItem method 1-8
- GetEffectiveItemNum method 1-8
- GetHistoryItems property 1-5
- GetKeyInfoCollection property 1-5
- GetPropertyByName method 1-8
- GetPropertyInfoByName method 1-7, 1-8
- getting ItemByKeys signature 2-18
- InsertItem method 1-7
- InteractiveMode property 1-5
- introduction 1-1
- Item method 1-7
- ItemByKeys method 1-7
- ItemByKeys signature 2-18
- java example 4-12
- keys 1-3
- keys, adding and removing 2-7
- methods 1-6, 2-8
- methods, data collection 1-8
- methods, DataRow 1-8
- methods, security for 1-8
- methods, standard 1-6, 2-8
- methods, user-defined 1-6, 2-9
- naming 1-3
- PeopleCode events and functions 3-5
- PeopleCode, client-only limitations 3-6
- properties and collections 1-4
- properties and collections, security for 1-5
- properties to expose 2-7
- properties, creating 2-5
- PropertyInfoCollection property 1-5
- removing keys 2-7
- search dialog processing 3-5
- security 2-10
- security for methods 1-8
- security for properties 1-5
- sending email 3-6
- SetPropertyByName method 1-7, 1-8
- standard methods 1-6, 2-8
- standard properties 1-4
- testing 2-13
- user-defined methods 1-6, 2-9
- user-defined properties 1-4
- validating 2-20
- WinMessage status 3-6
- component interface API 1-2
- ComponentName property 1-5
- components

- and component interface 1-2
- CopyRowsetDelta method 1-7
- create method 1-6
- CreateKeyInfoCollection property 1-5
- creating component interface 2-3
- creating component interface properties 2-5
- CurrentItem method 1-8
- CurrentItemNum method 1-8

D

- data collection for component interface 1-8
- DataRow methods for component interface 1-8
- DeleteItem method 1-7

E

- email
 - and component interface 3-6
 - sending with component interface 3-6
- extapi directory 3-3

F

- FindKeyInfoCollection property 1-5

G

- generating VB template in component interface 2-21
- GetEffectiveItem method 1-8
- GetEffectiveItemNum method 1-8
- GetHistoryItems property 1-5
- GetKeyInfoCollection property 1-5
- GetPropertyByName method 1-8
- GetPropertyInfoByName method 1-7, 1-8
- getting ItemByKeys signature 2-18

I

- InsertItem method 1-7
- InteractiveMode property 1-5
- Item method 1-7
- ItemByKeys method 1-7
- ItemByKeys signature 2-18

J

- java example
 - in component interface 4-12

K

- keys for component interface 1-3

M

- methods 1-6
 - data collection 1-8
 - DataRow 1-8
 - security for 1-8
 - standard 1-6
 - user-defined 1-6
- methods, standard 2-8
- methods, user-defined 2-9

N

- naming component interface 1-3
- naming conventions
 - component interface 1-3

O

- options
 - for COM type library area 3-1

P

- PeopleBooks
 - CD-ROM, ordering ii
 - printed, ordering ii
- PeopleCode
 - client-only limitations 3-6
 - component interface access 3-6, 4-2
 - generating for component interface 2-24
- PeopleCode events and functions
 - component interface 3-5
- PeopleSoft APIs
 - building for component interface 3-1
- properties
 - component interface 1-4
- properties and collections, security for 1-5
- PropertyInfoCollection property 1-5

S

- search dialog processing
 - component interface 3-5
- security
 - component interface 1-8
 - for component interface properties 1-5
- security for component interface 2-10
- security for component interface methods 1-8

SetPropertyByName method 1-7, 1-8
standard methods 2-8
standard methods for component interface 1-6
standard properties 1-4

T

testing
 component interface 2-13

U

user-defined methods 1-6, 2-9
user-defined properties 1-4

V

validating component interface 2-20
Visual Basic template
 example of file 2-21
 generating one for component interface 2-21

