Online Development

Student Handbook

ProBiller Online Development Student Handbook

Table of Contents

Chapter

1

Introduction

Overview

The ProBiller Online Development handbook has been designed to accompany the Online Development course. The handbook presents materials that will be touched upon in class but that are too detailed to be included on a slide presentation. When you take the ProBiller Online Development course, your instructor may ask you to refer to the handbook for examples of points made during the course or to further explain certain concepts.

After the course, you can use the handbook as a reference tool as you work on ProBiller development.

What You Need to Read

This Online Development Guide is both a tutorial and a reference guide. If your first exposure to this guide is through the Online Development course, follow your instructor's instructions for using the guide in class. After class, you may use the guide as a reference or refer to individual chapters to refresh your memory on completing specific tasks.

ProBiller Online Development Handbook Features and Conventions

The following terms and notational conventions are used throughout this guide:

- Key names match the name shown on most keyboards and are capitalized in brackets, such as <SHIFT>.
- Keystrokes joined with a plus sign (+) signify that you should hold down the first key and tap the second. For example, the keyboard shortcut to open the Windows 95 or NT Start button would be indicated as <CTRL>+<ESC>.
- Names of screens, dialog boxes, and entry fields are written in title case, for example, the File Open dialog box.
- Names of buttons and other screen elements are written as they appear on screen. For example, you may see "Click the OK button to accept your changes."
- Field labels, such as **HQ Acct ID**, are printed in bold.
- Words or characters that you actually type are indicated in **bold**.

- Variables that you type are bracketed and bolded, as in **make <servername>**.

- *Italic type* is used to indicate references to other components of the ProBiller documentation package, such as the *Call Manager User's Guide*.

- Procedures are shown in numbered lists with bold numbers, such as **1)** Press <TAB> to move to the next field.

- Notes, special instructions, or warnings are set off with an open-book bullet, and the text is italicized, as in the example below.

📖 *Notes are used to bring a topic to your attention when further thought is warranted.*

- Results of a series of steps are set off with a check-mark bullet, and the text is italicized, as in the example below.

✓ *Result explanations help you to understand what you should see on your screen as a result of having performed a series of steps.*

Chapter

2

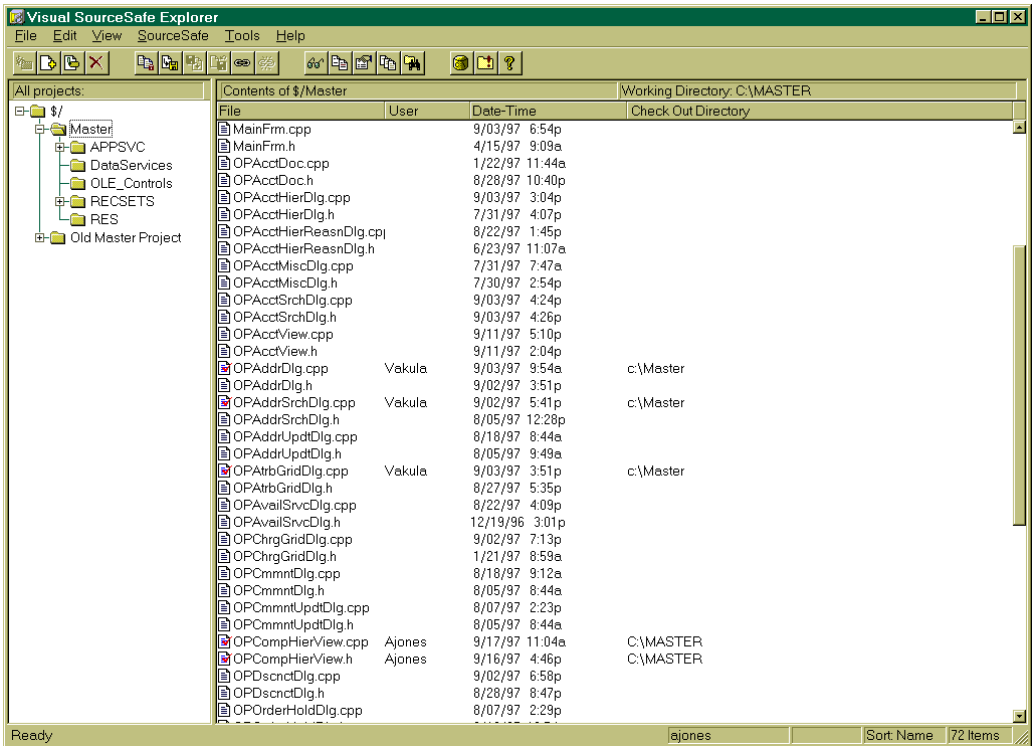
Directory Organization

Overview

This section describes the directory structure of ProBiller and provides you with a detailed look at the contents of each directory.

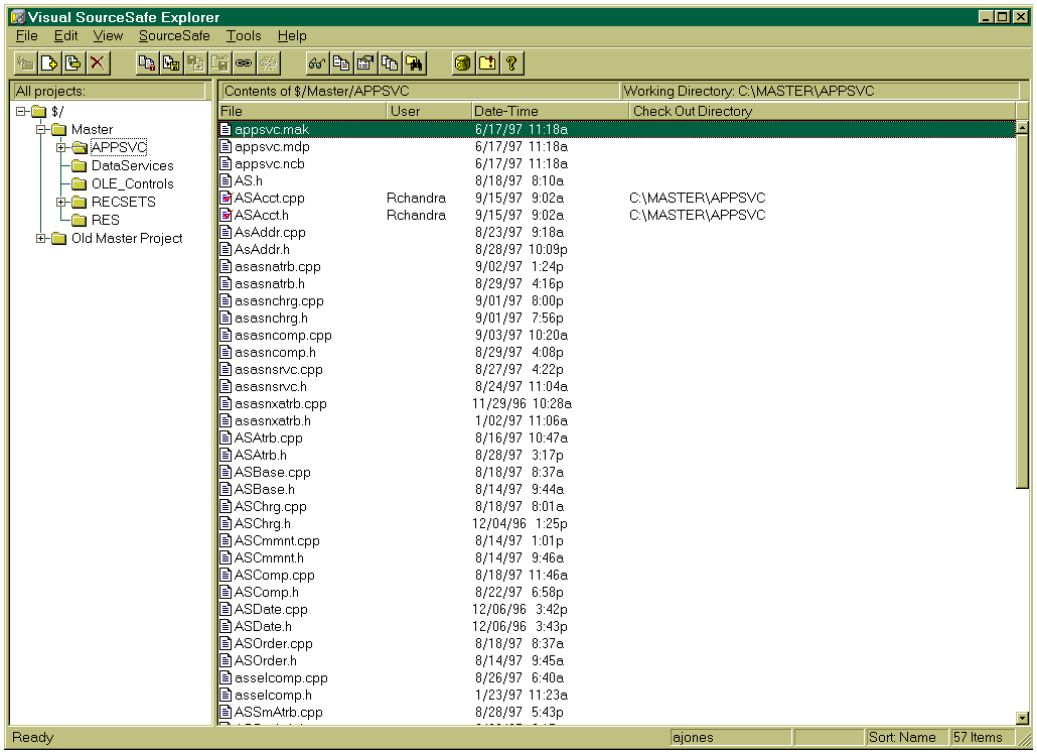
Master

The Master directory contains all of the programs that make up ProBiller.



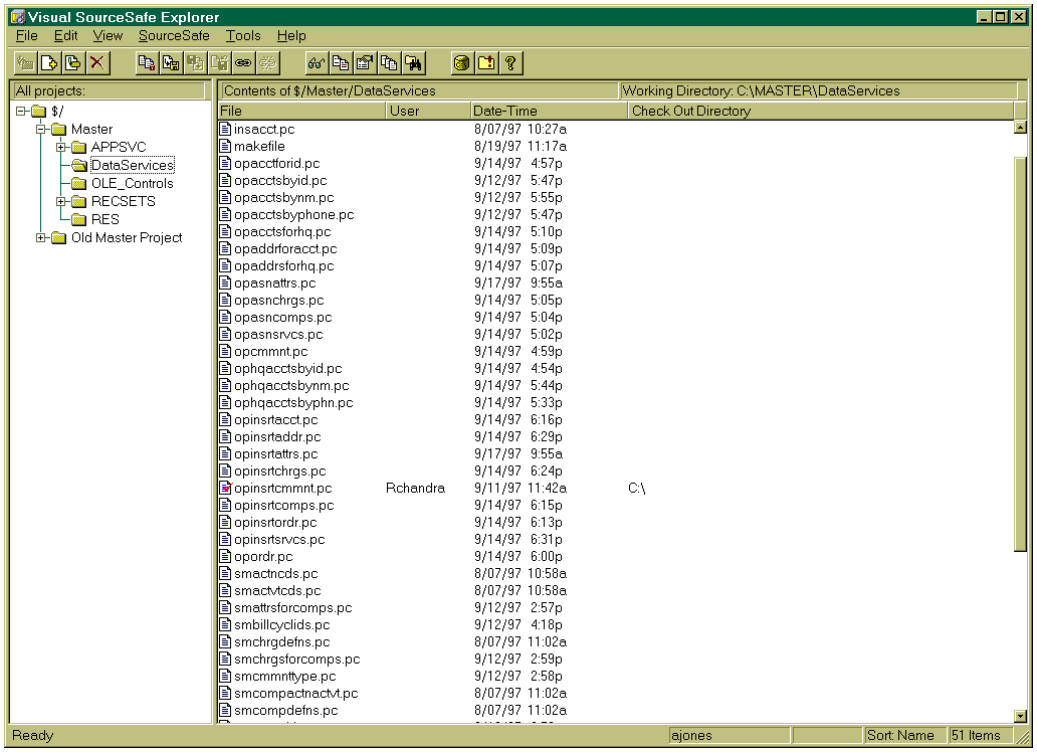
APPSVC

The APPSVC folder contains all of the application service classes.



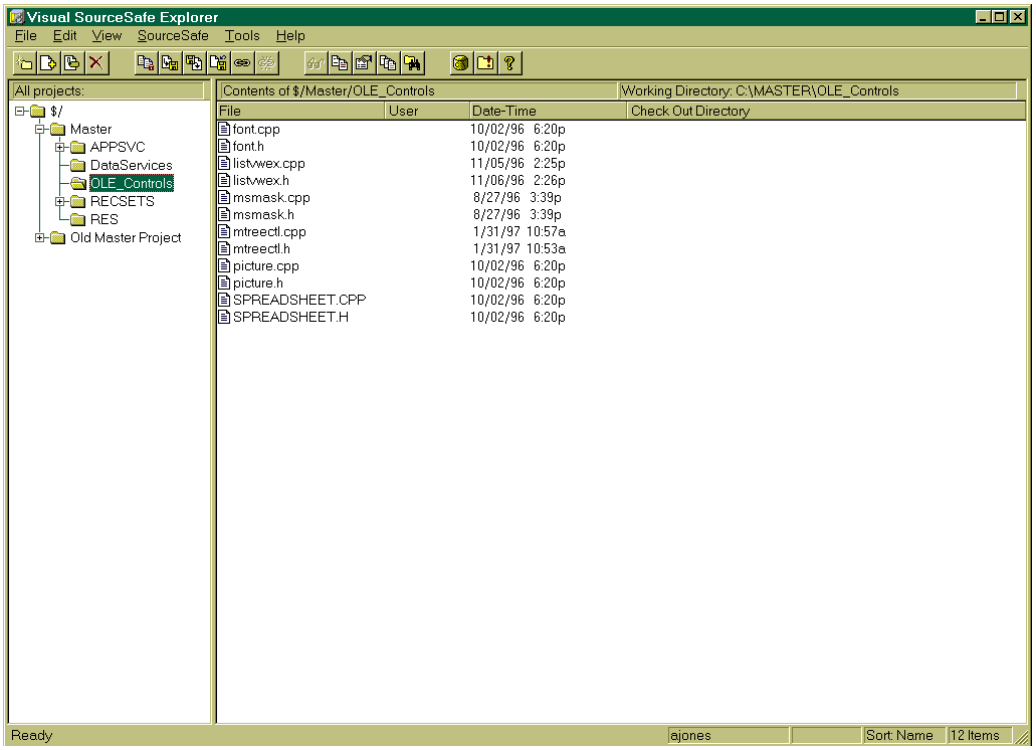
DataServices

The DataServices directory contains all of the Tuxedo services.



OLE_Controls

The OLE_Controls directory contains the OLE objects used in ProBiller development.



Chapter

3

Online Application Services

Overview

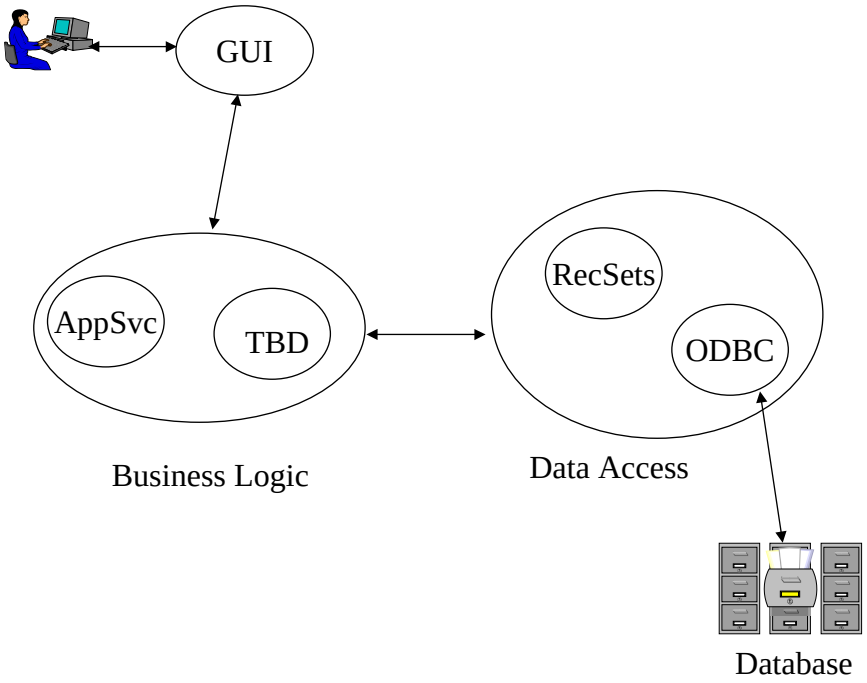
This section describes the ProBiller application services.

Introduction to the Application Services Library

The Application Services (AppSvc) library contains a collection of C++ classes used by the ProBiller application. The AppSvc classes model the entities associated with order processing, such as:

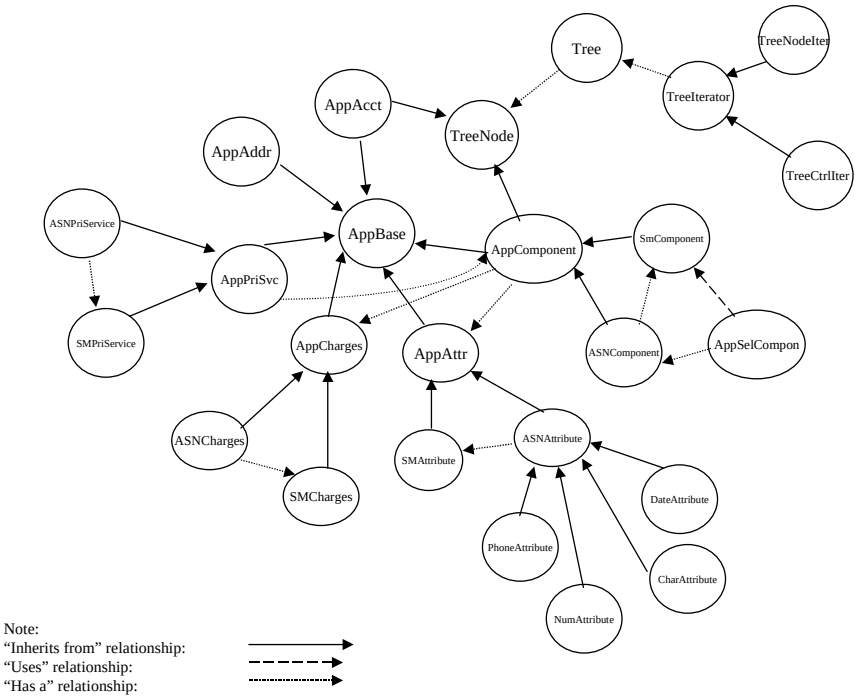
- Account
- Service
- Component
- Attribute
- Charges
- Order
- Address

Each AppSvc class contains the business logic applicable to the entity it represents and also maintains the relationship to other entities (i.e., Account <-> Service, Service <-> Component). AppSvc allows these entities to be saved into a database management system (DBMS) and retrieved from the DBMS for modifications. The AppSvc classes contain methods and attributes that an application program can use to perform business functions and to obtain and manipulate information for an object/entity. The following figure shows the relationship of AppSvc to other components of ProBiller.



AppSvc Hierarchy

The classes in AppSvc have the relationship shown in the following figure. Each AppSvc class is described in the succeeding paragraphs.



ASBase

ASBase is the root/base class from which all other AppSvc classes inherit. ASBase is a pure virtual class; that is, an AppBase object cannot be instantiated. ASBase contains functionalities common to all other AppSvc classes, such as the capability to perform database operations.

Each instance of the ProBiller application shall have just a single connection to the working database. ASBase maintains this single database connection as a static variable. The method *GetDBConn()* can be called to obtain this database connection. If a database connection doesn't yet exist when GetDBConn() is called, then GetDBConn() will automatically create the database connection. *The database that will be connected to by GetDBConn() is specified by the DBNAME key in the system Registry, or it can be specified from the command line*.

Tree Classes

Several ProBiller entities have a hierarchical relationship with other entities of its type. An account hierarchy is one such case. Two AppSvc classes exist to implement such a relationship, ASTree and ASTreeNode. An ASTreeNode object represents an item that can be inserted into an ASTree object, while an ASTree object contains a hierarchy of ASTreeNode objects.

Class ASTreeNode contains a list of sub-nodes as well as a pointer to its parent node to implement the hierarchical relationship. Each ASTreeNode object in a Tree must have a unique identifier. For an object to be inserted into an ASTree object, it must be derived from ASTreeNode. Two other AppSvc classes are derived from ASTreeNode, which are ASAcct and ASComp.

An ASTree object contains a map of ASTreeNode objects in the tree. When inserting a node into an ASTree object, the identifier of the parent node must be specified to indicate where the node is to be inserted. This requires that every node in the ASTree object must be uniquely identified and that the tree must be built in "hierarchical" order (i.e., a parent node must be inserted before a child node). To insert a node into an ASTree object, the method *Insert* is called. The input parameters for *Insert* specifies the node to insert and where within the tree to insert it. *Insert* looks up the "where" part in the map (a map is a container that allows for fast lookups and inserts) of the ASTree object. The method *Delete* is used to delete a node and its sub-nodes from the tree. To find a certain node in the tree, the method *Lookup* can be called.

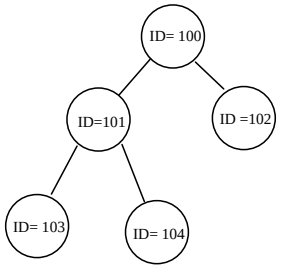
Accounts

The ASAcct class is used to represent a customer account. ASAcct inherits from ASBase and ASTreeNode. An AppAcct object can be saved to and retrieved from the database (ACCT table) by calling Load and Save. Class ASAcct contains an attribute for each field in the ACCT table.

The method *LoadHierarchy* can be called to load from the database an entire hierarchy of Account objects associated with a Headquarter account. LoadHierarchy applies a query such as the following SQL statement to the ACCT table to perform the retrieval:

Select * from ACCT where HQ_ACCT_ID = <Some Account ID>

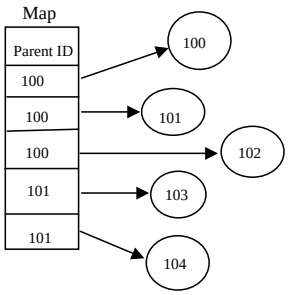
LoadHierarchy then performs a conversion of the array of records (recordset) returned from the query from a “relational” format to a “hierarchical” format. To perform the conversion, LoadHierarchy goes through each record returned by the query and builds an ASAcct object with the information in the record. LoadHierarchy inserts all the ASAcct objects into a map indexed by the parent account identifier (ACCT_PARENT_ID). To build a tree/hierarchy of account objects, LoadHierarchy creates a ASTree object. The ASAcct object representing the headquarter account is then inserted into the account tree. LoadHierarchy then looks in the map to find the ASAcct objects, whose parent is the headquarter account object, and inserts them into the account tree. This process continues until all account objects are inserted into the tree. The following figures illustrates the processing performed by LoadHierarchy.



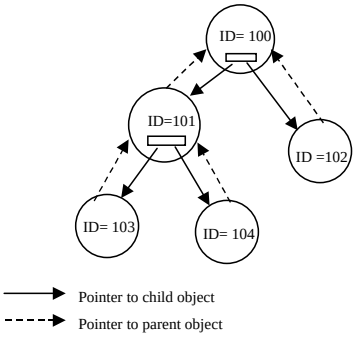
A. Logical view of an account hierarchy

Acct ID	HQ ID	Parent ID	...	
100	100	100	...	
101	100	100	...	
102	100	100	...	
103	100	101	...	
104	100	101	...	

B. Recordset returned from query of account hierarchy



C. ASAcct objects indexed by the parent account identifier



D. Hierarchy of ASAcct objects in memory.

After an account hierarchy has been established in memory, each account object in the hierarchy maintains pointers to its remittance account object and headquarter account object. An application can obtain this information by calling the methods *GetHeadQuarter* and *GetRemittance*.

An account can have several services assigned to it. To model this relationship, class ASAcct maintains a linked list of pointers to the assigned service objects (class ASSrvc). When a service is ordered for an account, the method *AddService* of an ASAcct object is called to add a service object to that ASAcct object. The method *RemoveService* does the opposite. The method *GetNumServices* returns the number of services assigned to a certain account. To iterate through all the services assigned to an account, the method *GetService* can be used.

When an account object is deleted from memory, the class destructor will initiate the removal from memory of any service objects assigned to that account object. This will start a chain of the memory cleanup of the service, component, attribute, and charges objects associated with that account.

Service Classes

The service entity is implemented by three classes:

- ASSrvc
- ASSmSrvc
- ASAsnSrvc

ASSrvc contains information common to ASSmSrvc and ASAsnSrvc, and is inherited by those two service classes. ASSmSrvc represents a system maintenance (SM) service entity. ASAsnSrvc represents a service entity that has been assigned to an account.

Class ASSrvc contains a pointer to a hierarchy of components. For a SM service, the components are those that can be assigned to that service. For an assigned service, the components are those that have been assigned to that service (and thus to the account).

Class ASSmSrvc contains an attribute for each field in the SRVC_SM table. Class ASSmSrvc contains a method, *LoadPrimaryComponentsSM*, that can be called to load all the SM services information from the database table SRVC_SM into ASSmSrvc objects. Because the SM services are static and do not change, these services are retrieved once from the database and kept in memory throughout the duration of the application. Subsequent calls to LoadPrimaryComponentsSM will not cause any retrieval to be performed. Another method, *LoadComponentHierarchySM*, is called to load all the SM components associated with a service. The process of loading a hierarchy of SM components resembles the process of loading an account hierarchy, described earlier. Similar to the SM services, the SM components need to be retrieved only once, and are kept in memory throughout the duration of the application.

When a service is ordered for an account, an ASAsnSrvc object is instantiated. An ASAsnSrvc object doesn't contain all the information about a service. Static information such as the service description and service type are stored in the ASSmSrvc objects. Thus, each ASAsnSrvc object must contain a pointer to its corresponding ASSmSrvc object.

An ASAsnSrvc object can be saved to and retrieved from the database (SRVC_ASN table) by calling Load and Save. An ASAsnSrvc object is only inserted or retrieved from the database when its assigned account is inserted or retrieved from the database.

An ASAsnSrvc object is removed from memory when its assigned account is removed from memory. The class destructor of ASAsnSrvc initiates the removal from memory of the hierarchy of components assigned to the service.

Component Classes

The component entity is implemented by three classes:

- ASComp
- ASSmComp
- ASAsnComp

Class ASComp inherits from ASBase and ASTreeNode. ASComp contains information common to ASSmComp and ASAsnComp, and is inherited by those two component classes. ASSmComp represents a system maintenance (SM) component entity. ASAsnComp represents a component entity that has been assigned to a service.

A component can have many attributes and charges associated with it. To implement this relationship, class ASComp contains a linked list of pointers to Attribute objects and a linked list of pointers to Charges objects. Methods exist in class ASComp to insert attributes and charges to a component and to iterate through the charges and attributes of a component. Both classes ASAsnComp and ASSmComp contain methods to load the associated attributes (LoadAttributes) and charges (LoadCharges) from the database.

The SM components are loaded from the database tables COMP_HIER_SM and COMP_DEFN_SM. The loading process of SM components is similar to the process described earlier for loading accounts. The SM components are loaded as part of a component hierarchy of a particular service. When loaded, each unique SM component is instantiated as an ASSmComp object and stored in memory in a map indexed by the component identifier. Even if a SM component is part of more than one component hierarchy, only one ASSmComp object shall be instantiated. This one ASSmComp object is referred to by the many component hierarchies to which it belongs.

When a component is ordered for a service, an ASAsnComp object is instantiated. An ASAsnComp object doesn't contain all the information about a component. Static information such as the component name, code, and mandatory flag are stored in the ASSmComp objects. Thus, each ASAsnComp object must contain a pointer to its corresponding ASSmComp object.

An ASAsnComp object can be saved to and retrieved from the database (ASN_COMP table) by calling Load and Save. An ASAsnComp object is only inserted or retrieved from the database when its assigned service is inserted or retrieved from the database.

An ASAsnComp object is removed from memory when its assigned service is removed from memory. The class destructor of ASAsnComp initiates the removal from memory of the attributes and charges assigned to the component.

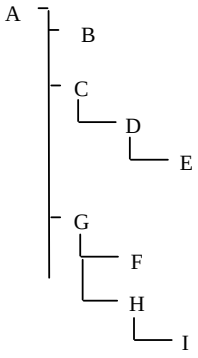
Ordering a Component

Class ASSelComp is used by the Component Hierarchy window to implement the functionality of component ordering. Class ASSelComp displays a hierarchy of SM components for a user to order and updates the hierarchy when a component has

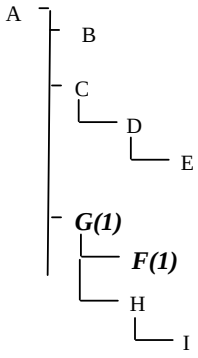
been ordered or deleted. Class ASSelComp uses the MFC class CTreeCtrl to display the hierarchy of components and also relies on the class ASAsnComp to implement assigned components.

In its initialization procedure, an ASSelComp object is instantiated by the OPCompHierView class, which implements the Component Hierarchy window. When ordering for a new service, any mandatory SM components are automatically ordered. The LoadASNComp method of ASSelComp is called by OPCompHierView to populate the tree control of the window with the SM components and any assigned components.

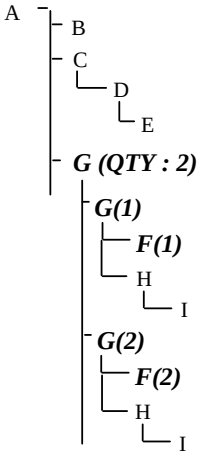
When a component is ordered, OPCompHierView calls the method *AssignComponent* to update the tree control with the new assigned component. The following figures show the visual updates to the tree control performed by AssignComponent when a component is ordered.



A. Hierarchy of SM components



B. Order component G.



C. Order second instance of component G.

Part A of the figure shows the component tree in its original state with no ordered components. In part B, component G was ordered. Assuming that component F is a mandatory component, it was also ordered. In part C, another G component had been ordered. Notice that a grouping label was added to the tree to indicate the quantity of G components, and that the SM components of H and I are displayed un-

der both instances of the assigned G components. The different instances of an ordered component are identified by their distinguishing attribute.

The method *AssignComponent* calls on two other procedures, LoadASNComp and LoadSMComp, to display the tree control with the SM and assigned components. The following pseudo-code reveals the logic implemented by LoadASNComp and LoadSMComp.

LoadASNComp

```
Display ASN component
Get corresponding SM component

For each child component of SM component
    Does ASN component have any matching sub components
        No:     Call LoadSMComp
        1:      Call LoadASNComp
        > 1:    Display SM Group Component (Group Label)
                For each ASN component
                        Call LoadASNComp
                Next
Next
```

LoadSMComp

```
Display SM component
For each child component of SM component
    Call LoadSMComp
Next
```

Internally, class ASSelComp maintains an ASTree object (or hierarchy) of assigned components. When a component is ordered, in addition to performing the visual update of the tree control, ASSelComp adds the assigned component to its internal ASTree object of assigned components. The method *DeAssignComponent* can be called to remove the assignment of a component.

When the order of components for a service is approved by the user, the ASTree object of assigned components maintained by the ASSelComp object is moved to the assigned service object.

Attributes

The attribute entity is implemented by the following classes:

- ASAtrb
- ASSmAtrb
- ASAsnAtrb
- ASAsnCharAtrb
- ASAsnPhoneAtrb

- ASAsnDateAtrb
- ASAsnNumAtrb

ASAtrb contains information common to ASSmAtrb and ASAsnAtrb, and is inherited by those two attribute classes. ASSmAtrb represents a system maintenance (SM) attribute entity. ASAsnAtrb serves as the base class for the classes ASAsnCharAtrb, ASAsnPhoneAtrb, ASAsnDateAtrb, and ASAsnNumAtrb, all of which represent a specific type of assigned attribute.

When an SM component is loaded from the database for the Component Hierarchy window, and that component is deemed mandatory for the service to which it belongs, then the SM attribute(s) for that SM component will be loaded (from the COMP_ATRB_SM table) also. Otherwise, in order to conserve system resources, SM attributes are only loaded from the database when requested by the user—that is, when a user clicks on an SM component in the Component Hierarchy window, the corresponding SM attributes are shown in the attribute grid.

When loaded, each unique SM attribute is instantiated as an ASSmAtrb object and stored in memory in a map indexed by the attribute code. Even if an SM attribute is a member of more than one component, only one ASSmAtrb object shall be instantiated. This one SM attribute is referred to by the many components to which it belongs. The SM attributes of an SM component are kept in memory for the duration of the application.

When an ASAsnComp object is created as a result of a component being ordered, ASAsnAtrb-derived objects are also created for that assigned component. The SM attributes of the corresponding SM component for the assigned component is used to determine what attributes must be created. The method *NewAsnAtrb* is used to create an assigned attribute object based on its corresponding SM attribute. If the format of the SM attribute is a telephone number, then an ASAsnPhone attribute object will be created. The other types of assigned attributes are created in a similar manner.

An ASAsnAtrb object does not contain all the information about an attribute. Static information such as the attribute name, component code, and mandatory flag are stored in the ASSmAtrb objects. Thus, each ASAsnAtrb object must contain a pointer to its corresponding ASSmAtrb object.

The ASAsnAtrb-derived classes contain methods that are used primarily by the Component Hierarchy window in dealing with the attribute grid. For example, calling the method *GetCellType* on an ASAsnPhoneAtrb object will return the grid cell type code to use for a phone number, while calling the same method on an ASAsnDateAtrb object will return the grid cell type code for a telephone number. The method *Validate* is used to verify that the attribute object contains a value that is appropriate for that type of attribute object.

An ASAsnAtrb-derived object can be saved to and retrieved from the database (ASN_ATRB table) by calling Load and Save. An ASAsnAtrb-derived object is only inserted or retrieved from the database when its assigned component is inserted or retrieved from the database.

An ASAsnAtrb derived object is removed from memory when its assigned component is removed from memory.

Charges

The Charge entity is implemented by the following classes:

- ASChrg
- ASSmChrg
- ASAsnChrg

Overall, the charge classes are structurally analogous to the attribute classes, differing in only that they represent different entities. Both ASSmChrg and ASAsnChrg objects have the same lifetime as their counterpart attribute objects; they are instantiated and destroyed at the same time.

Dates

The class ASDate is used by ProBiller to perform general date and time management. ASDate inherits from the MFC COleDateTime class. ASDate has the additional capability to obtain and parse a date in the formats useful to ProBiller, such as the following formats:

- YYYYMMDD (e.g., 1994/11/21)
- MMDDYYYY (e.g., 11/13/1994)
- MMDD (e.g., 10/23)
- MMYYYY (e.g., 12/1994)

When instantiated, an ASDate object shall contain the current day, month, and year. The method *SetDate* can be used to set a new date value into the object. The date string used as the input to SetDate must be in one of the formats specified above. If the date format and value are valid, ASDate will correctly parse the date string and assign the new date to the object.

To obtain the date value in an ASDate object in one of the above formats, the method *GetDate* can be called.

Date strings in different formats can be compared using ASDate (e.g., comparing 10/1994 against 10/24/1994) by using *CompareDate*.

Address

The ASAddr class is used to model an address entity. The AsAddr class is also used to support the Address window of the ProBiller application. An AsAddr object can be queried from and inserted into the database table ADDR. Class ASAddr contains an attribute for each field in the ADDR table.

The methods *GetLine1*, *GetLine2*, and *GetLine3* are used to return the address in a string format. GetLine1 returns the street number and name information concatenated into a string. GetLine2 returns other parts of the street address, such as any

dwelling unit number or post office number. GetLine3 returns the county, state, and zip code number information concatenated into a string.

The method *Validate* can be used to verify the content of an ASAddr object. Validate will perform some rudimentary checking for a valid state code and zip code, and will additionally check to ensure that other mandatory address components are populated.

Chapter

4

Online Coding Conventions

Overview

This chapter describes the online coding conventions in place for ProBiller developers.

Naming Conventions

File Names

File names should be designated with all capital letters. The following naming convention should be used for all source objects:

Format:	SNNNNNCT		
Where:	S	=	One character system code ('O'rder Processing, 'S'ystem Configuration, 'M'essage Processing, 'B'illing)
	NNNNN	=	Descriptive activity code (e.g., AADDR for Account Address, AMISC for Account Miscellaneous)
	C	=	Reserved character, defaulted to 'C'
	T	=	Class type

The valid Class types are:

A = Application Service (classes defined by Triet)
S = RecordSet
D = Dialog
F = Form
V = View

Examples

OAADDRCF for Order Processing's Account Address Form
OAMISCCD for Order Processing's Account Miscellaneous Dialog

Resource Identifier Names

Resources will be designated with all capital letters. The following naming convention should be used for all resources:

Format: XXX_CCC...

Where: XXX = Three character prefix assigned by Visual C++ for each resource type.

Valid resource types and their three character prefix:

Menu	=	IDR
Dialog	=	IDD
Bitmap	=	IDB
Icon	=	IDR
Toolbar	=	IDR
Controls	=	IDC

CCC... = Descriptive text for the resource with underscores
Note: Use the Standard Abbreviation List for the descriptive text.

Examples:

IDD_ACCT_PRF for Account Profile dialog
IDB_ACCT for Account Bitmap
IDC_ACCT_PRF_ BILL_CYCL for Bill Cycle control on Account Profile dialog

Naming Convention Rules

This section provides rules to which you should adhere when naming ProBiller data elements.

🕮 *The abbreviation list should be used for naming all variables, with the exception of Name and Date.*

Resource Identifiers

The following rules apply when naming controls, such as push buttons, edit boxes, group boxes, and edit masks.

Naming Convention	Example
"ID" (Identifier) should always be in upper-case.	AcctID
"CD" (Code) should always be in upper-case.	DlvryCD
Keep the type of control in the control's ID.	IDC_EDIT..., IDC_TAB..., IDC_MASKED...

Class Variables

Class variables may be used in one or more functions for a given class.

Naming Convention	Example
Class variable should have the following format: **m_VariableName**. Exception: RecordSet Class.	m_UserName

Local Variables

Unlike class variables, there is no standard prefix for a local variable. The scope of a local variable does not extend beyond the function in which it is defined.

Naming Convention	Example
Local variables should be descriptive and should have the following format: **VariableName**.	UserName

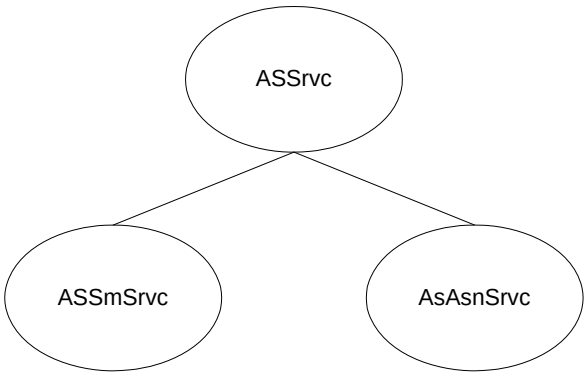
Control Variables

Control variables are used for elements such as list controls, combo boxes, and tree controls. Control variables always have a prefix of **m_ctl**.

Naming Convention	Examples
Control variables should have the following format : **m_ctlVariableNameContrlType**	m_ctlBillCycleCombo (Combo Box) m_ctlAcctSrchList (List Control) m_ctlCompHierTree (Tree Control)

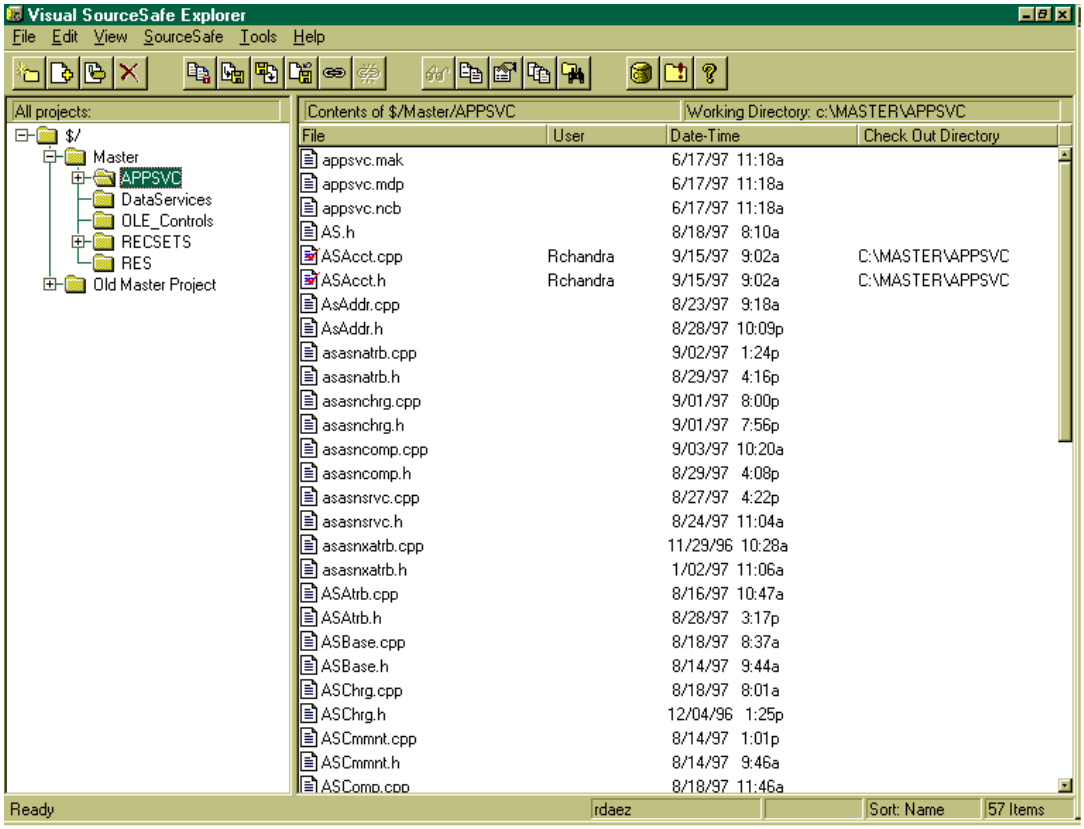
Naming Conventions for Application Services

Each application service encapsulates functions for a specific entity. Classes that deal with entities that are not derived from system maintenance (SM) tables should follow the naming format **ASEntityAbbreviation**, such as ASAcct. Some entities, such as ASSrvc, function as base classes from which other classes can be derived. For example, two classes are derived from the base class ASSrvc: ASSmSrvc and AsAsnSrvc, as illustrated below.



Naming Convention	Examples
Application services should have the following format: **ASVariableName**. After the AS prefix, which stands for *Application Services*, use ProBiller standard abbreviations (see Standard Abbreviation list) to name the service type.	ASAsnAtrb.cpp (Assign Attribute) ASComp.cpp (Component) ASOrder.cpp (Order)

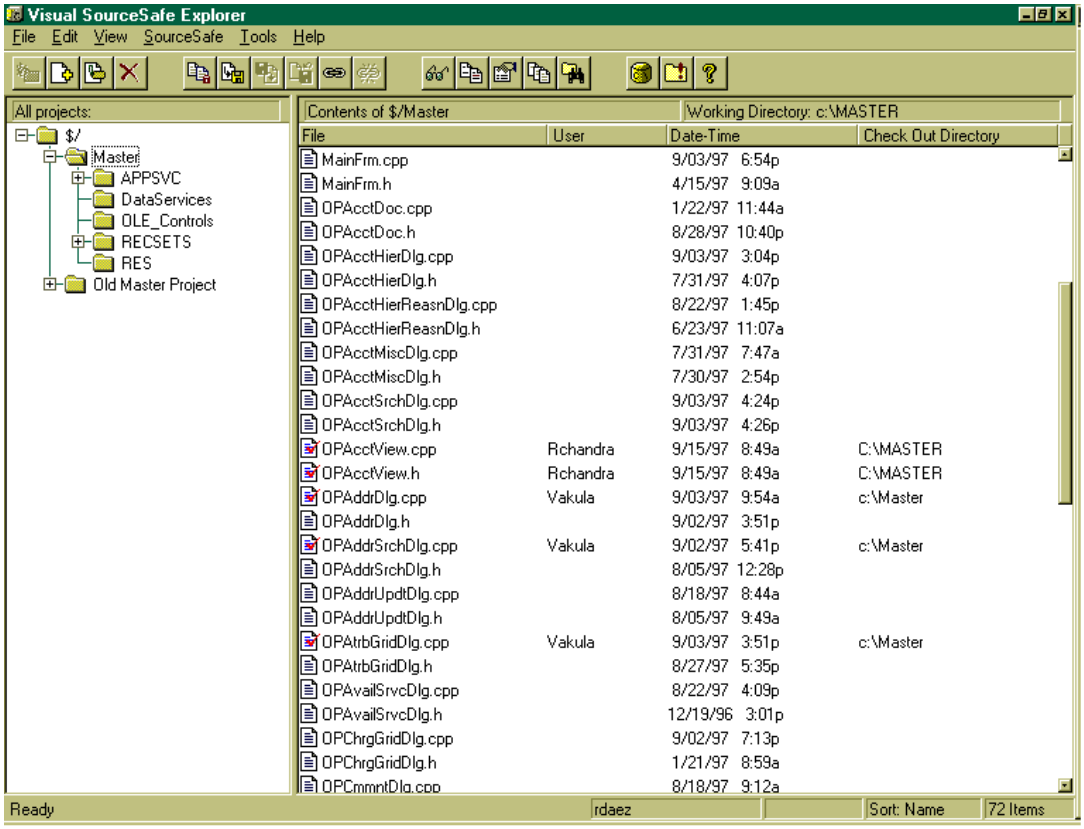
The Application Services Directory



Naming Conventions for Presentation Layer Classes

Naming Convention	Examples
Presentation layer classes for order processing must have the following format : **OPVariableNameClassType**. After the OP prefix, use ProBiller standard abbreviations (see Standard Abbreviation list) to name the presentation layer type.	OPAcctHierDlg.cpp (Account Hierarchy Dialog), where OP = Order Processing AcctHier = variable name Dlg = class type OPCmmntUpdtDlg.cpp (Comment Update Dialog) OPSrvcDlg.cpp (Service Dialog)
Presentation layer classes for system maintenance must have the following format: **SMVariableNameClassType**.	SMLogonDlg.cpp
Basic classes that are instantiated for each project do not have a prefix.	Summit.cpp MainFrm.cpp
Third party controls retain their original names.	Font.cpp MSMask.cpp Spreadsheet.cpp

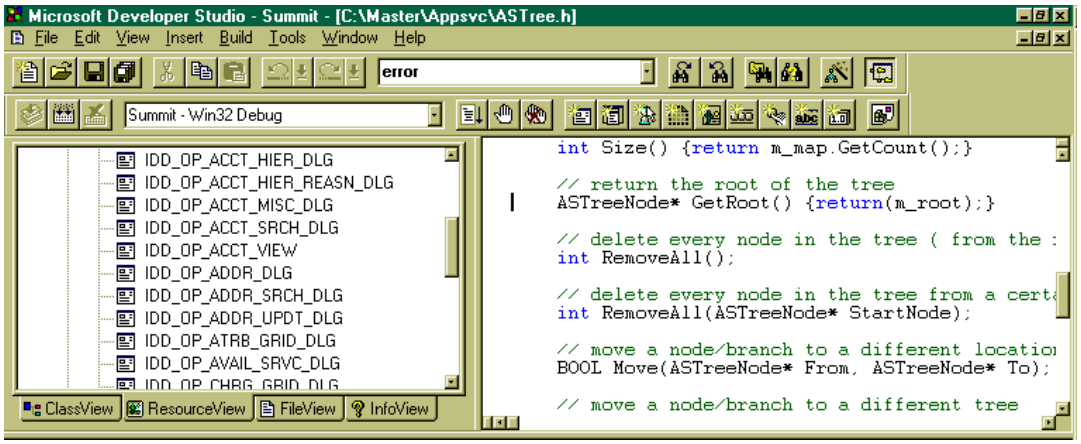
The Presentation Layer Classes Directory



Naming Conventions for Dialogs

Naming Convention	Examples
Dialogs should have the following format: **IDD_SUBSYSTEM_VARIABLENAME _CLASSTYPE**. Note that each element of the dialog name is separated with an underscore, and the **variable-name** part of the dialog name could contain underscores, as in IDD_OP_ACCT_SRCH_DLG.	IDD_OP_ACCT_VIEW (Account View) IDD_SM_LOGON_DLG (Log on dialog)

The Dialog Directory



Program Headers

General Header Information

Each window in ProBiller consists of two files—a **.h** file, which includes only a header, and a **.cpp** file.

At the beginning of every header, include the following information:

- Header revision
- Descriptive name of the dialog or view

Example

```
SMLogonDlg     -> ProBiller Logon Dialog
OPCompHierView -> Component Hierarchy View
_____________ Forward Header _____________________
Subject:  Headers for include files and handlers
Author:   AJONES at ICSPO1
Date:     12/2/96 11:12 AM
```

Include Headers in .h Files

Include the following information in the header of **.h** files.

```
//////////////////////////////////////////////////////////////////
// System:
// Subsystem:
// Name:
// Description:
// Inputs:
// Outputs:
//////////////////////////////////////////////////////////////////
```

Note on Include Headers

- Information after the colons in the Include Header should begin in column 17.

Function Headers in .cpp Files

Use the following format for each header.

```
//////////////////////////////////////////////////////////////////
// Description:
// Inputs:
// Outputs:
// Return Value:
//////////////////////////////////////////////////////////////////
```

📖 *Each function within a **.cpp** file has its own header. Therefore, there may be multiple headers within each **.cpp** file.*

Notes on Function Headers

- Information after the colons in the Function Header should begin in column 21.
- Place the Function Header below the name of the function.
- If a function has no inputs, outputs, or return value, type **N/A**.

General Header Note

- Slashes (//) and other header information stop at column 80.

Sample Header

```
////////////////////////////////////////////////////////////////
// System:       ProBiller
// Subsystem:    System Maintenance
// Name:         SMLogonDlg
// Description:  This dialog verifies the User Name and Password
//               and determines if the user has authorization to
//               use the ProBiller application.
// Inputs:       N/A
// Outputs:      N/A
////////////////////////////////////////////////////////////////
```

Program Headers

This chapter outlines the program headers that should be used in ProBiller online development.

General Header Information

Each window in ProBiller consists of two files—a **.h** file, which includes only a header, and a **.cpp** file.

At the beginning of every header, include the following information:

- Header revision
- Descriptive name of the dialog or view

Example

```
SMLogonDlg     -> ProBiller Logon Dialog
OPCompHierView -> Component Hierarchy View
_____________ Forward Header _____________________
Subject:  Headers for include files and handlers
Author:   AJONES at ICSPO1
Date:     12/2/96 11:12 AM
```

Include Headers in .h Files

Include the following information in the header of **.h** files.

```
//////////////////////////////////////////////////////////////
// System:
// Subsystem:
// Name:
// Description:
// Inputs:
// Outputs:
//////////////////////////////////////////////////////////////
```

Note on Include Headers

- Information after the colons in the Include Header should begin in column 17.

Function Headers in .cpp Files

Use the following format for each header.

```
////////////////////////////////////////////////////////////
// Description:
// Inputs:
// Outputs:
// Return Value:
////////////////////////////////////////////////////////////
```

📖 *Each function within a **.cpp** file has its own header. Therefore, there may be multiple headers within each **.cpp** file.*

Notes on Function Headers

- Information after the colons in the Function Header should begin in column 21.
- Place the Function Header below the name of the function.
- If a function has no inputs, outputs, or return value, type **N/A**.

General Header Note

- Slashes (//) and other header information stop at column 80.

Sample Header

```
////////////////////////////////////////////////////////////
// System:       ProBiller
// Subsystem:    System Maintenance
// Name:         SMLogonDlg
// Description:  This dialog verifies the User Name and Password
//               and determines if the user has authorization to
//               use the ProBiller application.
// Inputs:       N/A
// Outputs:      N/A
////////////////////////////////////////////////////////////
```

Chapter

5

Common Routines

Overview

This chapter describes the common routines available to ProBiller developers.

Common Routines

Account Search Routine

OPAcctSrchDlg

The Account Search routine is a modeless dialog that allows the user to search for accounts. The Account Search routine is activated under the following circumstances:

- When the user opens an account by selecting Account…Open
- When the user clicks on the HQ Acct ID search push button in the Account View to assign an account to a Headquarter Account
- When the user selects the Account Hierarchy button on the ProBiller button bar

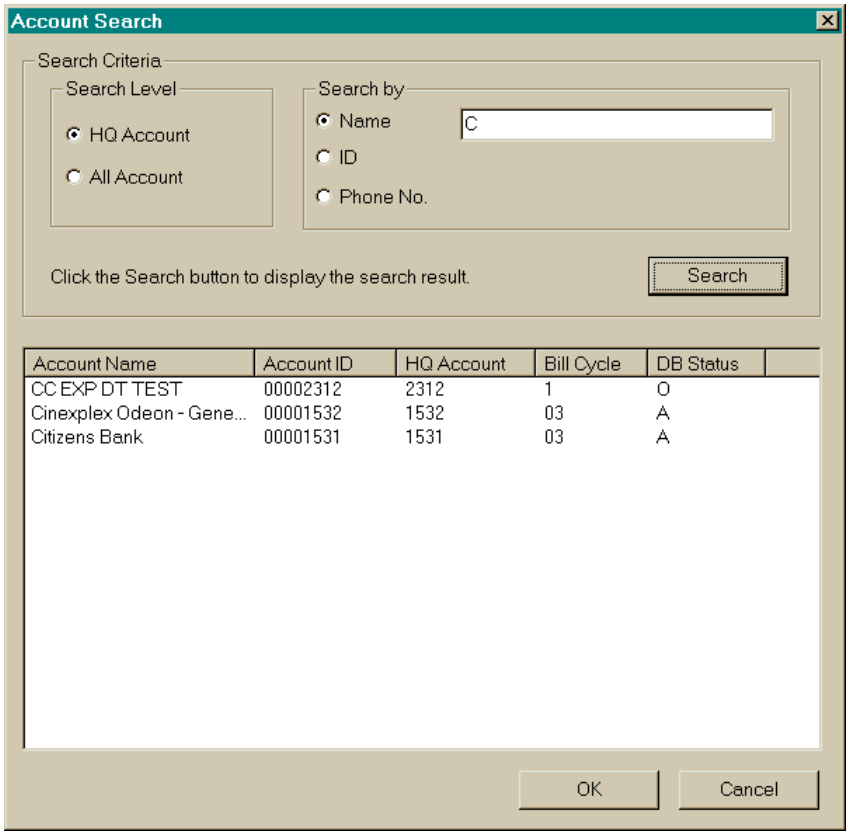
The Account Search dialog consists of three groupings—Search Level, Search By, and a List View control. There are two search levels for the Account Search dialog—Headquarter Accounts (where only Headquarter Accounts are displayed) and All Accounts. These search levels are displayed as option button controls. Headquarter Accounts is the default search level.

Accounts are sorted by the Account Name (Name) or Account ID (ID). These two criteria are represented by option buttons. The Name option button is the default for the Search By field. If the user wishes to start the account search from a particular account ID or Name, the *Type search value* field is used to enter the ID or name on which the search is to be performed.

Once the search criteria have been determined, the user can select the Search button, and accounts that meet the search criteria will be listed in the Account Search List View window. The user selects one of the accounts by clicking on the account and then selecting OK, or by double-clicking on the account name. The ID, Name, and Billing Cycle of the selected account are passed to the Account View and are displayed in their respective edit controls.

Users can select a search level of All Accounts or Headquarter Accounts and perform searches by name, ID, or telephone number.

The Account Search Dialog



Address Routine

OPAddrSrchDlg

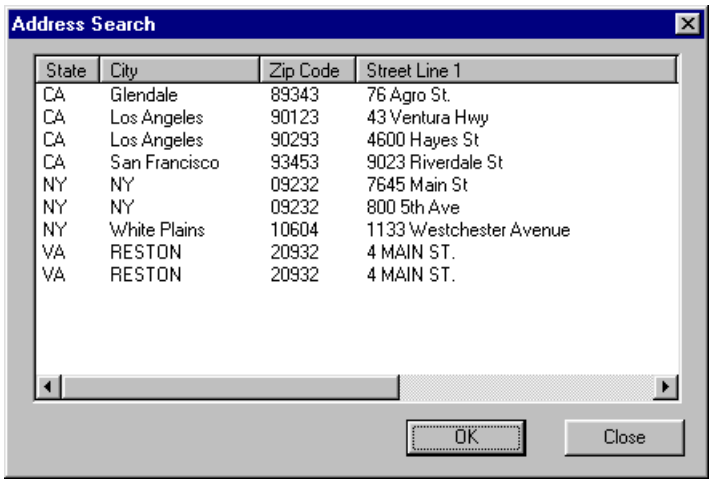
The Address Search dialog is called with the Search button from within the Address tab of the Account View window and from within the Remittance tab of the Account view window.

The Address Search Dialog is used to display the addresses of all accounts within a corporate structure. The Headquarter Account ID is passed in from the Account View. The Address Search dialog data is directly queried from the Address table.

The Address Search window consists of a List View and the OK and Cancel command buttons.The address information is displayed in the List View. The column headers for the List View are State, City, Zip Code Street Line 1 (Street Number and Name), Street Line 2 (Suite or Floor number) and Address ID.

Address information can be retrieved from the Address Search window in one of two ways. The user can either select one of the addresses from the address list and then select the OK command button, or simply double click on an address. Both actions will assign the address information from the Address Search dialog to the calling dialog and close the Address Search window. The Cancel button will simply close the Address Search window without passing any information.

The Address Search Dialog



OPAddrUpdtDlg

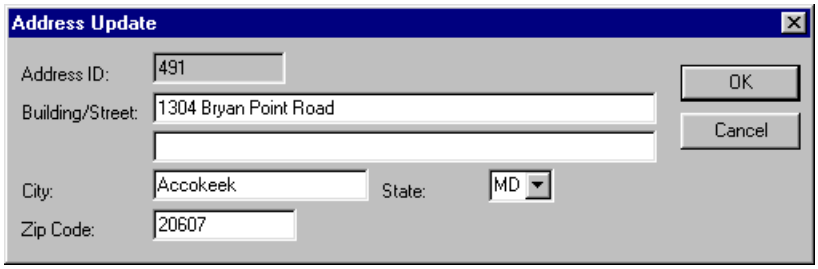
The Address Update dialog is used to add a new address or modify an existing address. It is called with the New or Edit buttons from within the Address tab of the Account View window and from within the Remittance tab of the Account View window.

The Address Update dialog window consists of the Address ID, Building/Street, City, and Zip Code edit controls. The State combo box data is directly queried from the System Maintenance State table.

The Address Update Dialog's edit controls are all empty when adding a new address. When modifying an existing address, the Address Update Dialog's edit controls are populated with the address information that was passed from the calling window.

The new address or modified address is returned to the calling dialog by clicking the OK command button. The user is not allowed to exit the Address Update Dialog with the OK command button if all mandatory fields have not been populated. The Cancel button will simply close the Address Update window without updating any information.

The Address Update Dialog



Disconnect Routine

OPDscnctDlg

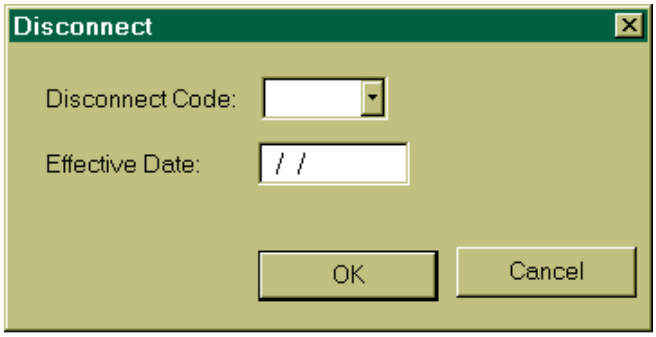
The Disconnect routine is a modeless dialog used to disconnect an account, service, or one or more components. It is called from within the Component Hierarchy View when the user selects the Disconnect command button.

Users select a disconnect code with the Disconnect Code drop-down dialog and type a valid start date into the Effective Date entry field. When the user selects the OK command button, the Disconnect dialog passes the user-selectable disconnect code and date to the Disconnect code table. If a user selects the Cancel command button, the Disconnect dialog closes without updating any information.

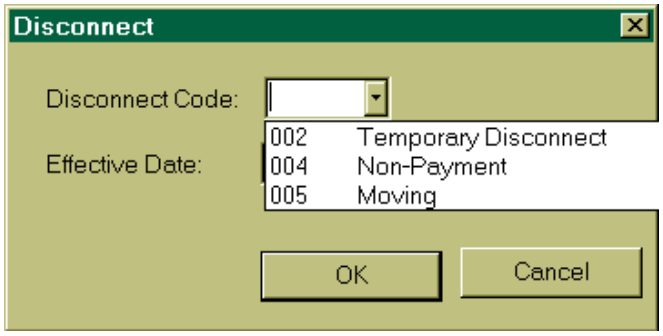
Currently, valid disconnect codes include Temporary Disconnect (002), Non-Payment (004), and Moving (005).

The two figures below picture the Disconnect Code dialog and the same dialog displaying the current Disconnect Code options.

The Disconnect Dialog



The Disconnect Dialog Displaying Valid Disconnect Code Reasons



System ID Routine

The System ID routine is called to generate a unique ID for an account, service, address, component, or charge.

The same routine is used regardless of which ID is being generated.

ASTree

The ASTree routine is used to load the account hierarchy routine and manipulate the account hierarchy structure.

The ID code within the system ID table is used to determine which entity to generate the ID for.

a

Chapter

6

Standard Abbreviations

Overview

This handout provides you with a complete alphabetized list of the standard abbreviations used by ProBiller developers.

Refer to the list of abbreviations when reading or creating table names, table fields, source code, flow charts, diagrams, or other ProBiller-related data structures.

Abbreviation	Meaning
ABBR	Abbreviation
ACC	Access
ACCT	Account/Accounting
ACD	Authcode
ACPT	Accept/Acceptable
ACT	Activity
ACTL	Actual
ACTN	Action
ADD	Additional
ADDR	Address
ADJ	Adjusted/Adjust/Adjustment
ADT	Audit
AGNT	Agent
AGR	Aggregate
ALGM	Algorithm
ALLOC	Allocation
ALLOW	Allowable
ALT	Alternate/Alternative
AMT	Amount
ANN	Annual
APP	Applies
APPL	Application
APPRV	Approve/Approved/Approval
AR	Accounts Receivable

Abbreviation	Meaning
ARCH	Archive
ASN	Assigned
ASSOC	Associated/Associates/Association
ASTD	Assisted
ATRB	Attribute
AUTH	Authorized/Authorization
AUTO	Automatic
AVAIL	Available
AVG	Average
BED	Begin Effective Date
BEG	Begin/Beginning
BILL	Billed/Billing/Billable
BLK	Block/Blockage
BRE	Billable Rate Element
BTN	Billing Telephone Number
BU	Backup
BUS	Business
CACCT	Corporate Account
CAL	Calendar
CAMP	Campaign
CAR	Carrier
CAT	Category
CC	Credit Card
CD	Code
CERT	Certificate
CHAR/CHR	Character/Characteristic
CHNG	Change
CHNL	Channel
CHRG	Charge
CIC	Carrier Identification Code
CL	Call
CLS	Close
CMMNT	Comment
CNCL	Cancel/Cancellation
CNT	Count/Counter
CNTR/CTR	Center

Abbreviation	Meaning
CNTRBTN	Contribution
CNTY	County
CO	Company
COLCT	Collect/Collection
COMM	Commission
COMP	Component
CONF	Conference
CONFG	Configuration
CONN	Connection
CONS	Consolidated
CONT	Contract
CONV	Conversion/Converted
COORD	Coordinate
CORP	Corporation
CORR	Correspondence/Corresponding
CPLT	Complete/Completion
CRATE	Charge Rate Schedule
CRCT	Circuit
CRDT	Credit
CRE	Creation
CRIT	Criteria
CST	Cost
CT	Customer Type
CTRL	Control
CTRY	Country
CUR	Current
CUST	Customer
CYCL	Cycle
DAYSV	Daylight Savings
DB	Database
DCHARVAL	Default Character Value
DDCTD	Dedicated
DEACT	Deactivate
DEF	Default
DEP	Dependency/Dependent
DESC	Description

Abbreviation	Meaning
DEST	Destination
DET	Detail
DF	Discount Factor
DICT	Dictionary
DIGT/DGT	Digits
DISCN	Disconnect/Disconnection
DISP	Disposition
DISQ	Disqualifier
DIV	Division
DLRS	Dollars
DLVR	Delivery
DMSC	Domestic
DNUMVAL	Default Numeric Value
DOC	Document
DOW	Day of Week
DPCT	Depiction
DRATE	Discount Rate Schedule
DRPD	Dropped
DSCT	Discount
DT	Date
DTCD	Detected
DTRMN	Determine/Determination
DUR	Duration
DVAL	Default Value
EC	Exchange Carrier
EED	End Effective Date
EFF	Effective
ELMT	Element
EMP	Employee/Employees
EMPR	Employer
END	Ending
ENTR	Entry
ENTRD	Entered
ENTY	Entity
EQUIP	Equipment
ERR	Error

Abbreviation	Meaning
EST	Eastern Standard Time
ESTB	Establish/Establishment
ESTM	Estimate/Estimated
EVNT	Event
EXC	Exception
EXEC	Execute/Executive
EXMPT	Exempt
EXP	Expiration
EXPCT	Expect/Expected
EXSTNG	Existing
EXT	Extract
EXTN	Extension
FCL	Facility
FCTR	Factor
FIL	File
FLD	Field
FLFLMNT	Fulfillment
FMT	Format
FNL	Final
FREQ	Frequency
FRN	Foreign
FST	First
FWD	Forward/Forwarding
GEN	General
GEO	Geographic/Geographical
GL	General Ledger
GOV	Government
GRP	Group
GRS	Gross
GRTOT	Grand Total
HAND	Handling
HCOORD	Horizontal Coordinate
HIER	Hierarchy
HL	Home LATA
HLDAY	Holiday
HORZ	Horizontal

Abbreviation	Meaning
HRS	Hours
HST	History
IC	Inter-exchange Carrier
ID	Identification/Identifier
IEC	Inter Exchange Carrier
IMP	Implementation
INC	Incorporated
IND	Indicator
INFO	Information
INIT	Initial
INITN	Initiation
INST	Install
INT	Interval
INTL	International
INV	Inventory
INVC	Invoice
INVST	Investigation
LAS	List Available Subroutine
LATA	Local Access and Transport Area
LEC	Local Exchange Carrier
LEN	Length
LIB	Library
LL	Lower Limit
LLV	Lower Level Value
LMT	Limit
LOC	Location
LOGIC	Logic/Logical
LOW	Lower
LST	Last
LVL	Level
MAIL	Mail/Mailing
MAJ	Major
MAN	Mandatory
MAT	Maturity
MAX	Maximum
MDA	Media

Abbreviation	Meaning
MDL	Model
MEAS	Measure
MEM	Member
MFR	Manufacture
MGMT	Management
MILE	Mileage
MIN	Minimum
MINS	Minutes
MKT	Market
MKTSG	Market Segment
MLSTN	Milestone
MNGR	Manager/Managerial
MNR	Minor
MNT	Month
MNTLY	Monthly
MOD	Modified
MSG	Message
MTHD	Method
MTR	Metered
MULT	Multiple
NB	Non-Switched Billing System
NBRPTS	Non-Switched Billing System Reports
NM	Name
NME	Non-Metered Event
NOLIS	Non-Listed
NOPUB	Non-Published
NPA	Phone Number Area Code
NTWRK	Network
NUM	Number
NXX	First Three Digits of Phone Number
OBT	Obtain
OCCR	Occurrence
OCCUP	Occupation
OE	Order Entry
OF	Office
OFCER	Officer

Abbreviation	Meaning
OL	Other LATA
OM	Operations Management
ONLN	Online
OP	Order Processing
OPER	Operations
OPRD	Operand
OPRPT	Order Processing Reports
OPT	Option/Optional
OPTR	Operator
ORD	Order
ORG	Organization
ORIG/OG	Original/Originating/Origination
OTH	Other
OVLP	Overlap
OVRD	Override
OVRL	Overall
PARTN	Partition
PBILL	Pre-Bill
PCST	Price/Cost
PCT	Percentage/Percent
PDEF	Pre-defined
PDTRMN	Pre-determined
PEN	Penetration
PER	Period
PGM	Program
PHN	Phone
PKG	Package
PL	Private Line
PLCMT	Placement
PLN	Plan
POS	Position
PR	Provisioning Request
PRCNG	Pricing
PRE	Prefix
PRELIM	Preliminary
PREV	Previous

Abbreviation	Meaning
PRF	Profile
PRMRY	Primary
PRMTN	Promotion
PRMTR	Parameter
PRN	Print
PRNT	Parent
PROC	Processing/Process
PROCD	Procedure
PROD	Product
PROMO	Promotion
PROP	Proposal
PRR	Prior
PRVDR	Provider
PSUB	Pre-subscription
PSWD	Password
PTH	Path
PYMT	Payment
QL	Qualification
QTY	Quantity
QUAL	Qualifier
RASN	Reassign/Reassignment
RC	Rate Characteristics
RCPT	Recipient/Receipt
RCVD	Received
RDO	Report Default Option
REC	Record
RECNC	Reconcile/Reconciliation
RECUR	Recurring
REF	Reference
REGN	Region
REL	Related
RELS	Release
REM	Remarks
REMIT	Remittance
RENDRNG	Rendering
REORG	Reorigination

Abbreviation	Meaning
REP	Representative
REQ	Request/Requested
RESN	Reason
RESOL	Resolution
RESP/RSP	Responsible/Responsibility
RESR	Resource
RESV	Reserved/Reservation
REV	Revenue
RLTNSHIP	Relationship
RO	Report Option
RPROC	Re-Processed
RPT	Report
RPTNG	Reporting
RQ	Required/Requirement
RSDNTL	Residential
RSPNS	Response
RT	Rate
RTE	Route
RTN	Routine
RVS	Revised/Revision
RVSL	Reversal
SA	Service Agreement
SAV	Savings
SB	Switched Billing
SBMT	Submit/Submitted
SBTOT	Subtotal
SC	Sales Commission
SCHED/SCH	Schedule
SCND	Second
SCRN	Screening
SCSFL	Successful/Success
SECNDRY	Secondary
SEL	Selection
SELL	Selling
SENS	Sensitive/Sensitivity
SEQ	Sequence

Abbreviation	Meaning
SGM	Segmented
SGNTR	Signature
SHRD	Shared
SIC	Standard Industry Code
SL	Service Location
SLS	Sales
SM	System Maintenance
SMRPT	System Maintenance Reports
SN	Special Number
SOFT	Software
SPCFC	Specific
SPCL	Special
SPHAND	Special Handling
SR	Subroutine
SRC	Source
SRCHG	Surcharge
SRL	Serial
SRVC/SV	Service
SSN	Social Security Number
ST	State
STAT	Status
STCS	Statistics
STD	Standard
STN	Station
STR	Start/Started
STRAD	Straddle/Straddling
STRT	Street
STRU	Structure
SUB/SUBSCRP	Subscription
SUD	Start-Up Date
SUFF	Suffix
SUM	Summary/Summarization
SUP	Supplied
SUS	Suspense
SW	Switch
SYS	System

Abbreviation	Meaning
TARF	Tariff
TBL	Table
TELE	Telephone
TEMP	Temporary
TERM/TER	Terminating/Termination
THRU	Through
TITL	Title
TKN	Taken
TM	Time
TMPLT	Template
TOT	Total
TR	Target Record
TRAN	Transaction
TREV	Total Revenue
TRGT	Target
TRK	Trunk
TRSH	Threshold
TRVL	Travel
TX	Tax/Taxing
TXBL	Taxable
TXID	Tax Identification
TYP	Type
UL	Upper Limit
ULV	Upper Level Value
UNIV	Universal
UNRND/URND	Unrounded
UNSFL	Unsuccessful
UNT	Unit
UP	Upper
UPD/*UPDT*	Update
URATE	Usage Rate Schedule
USG	Usage
VAL	Value
VAR	Variable
VARI	Variance
VCOORD	Vertical Coordinate

Abbreviation	Meaning
VERS	Version
VERT	Vertical
VLDTN	Validation
VNDR	Vendor
VOL	Volume
VSAM	Virtual Sequential Access Method
VTRAN	Verbal Translation
W/	With
WAI	Wait
WIR	Wire
WK	Work
WOL	Write Off Line
WTN	Working Telephone Number
XPN	Expense
XREF	Cross Reference
XTRCT	Extract
YR	Year

Chapter

7

ProBiller V3 Dialogs

Class Name	Class Description	Dialog Name	New/ Exists
Order Processing			
OPAcctDscntDlg	Account Disconnect Dialog	IDD_OP_ACCT_DSCNT_DLG	New
OPAcctHierDlg	Account Hierarchy Dialog	IDD_OP_ACCT_HIER_DLG	Exists
OPAcctMiscDlg	Account Miscellaneous	IDD_OP_ACCT_MISC_DLG	Exists
OPAcctHierReasnDlg	Account Reassign Dialog	IDD_OP_ACCT_HIER_REASN_DLG	New
OPAcctSrchDlg	Account Search Dialog	IDD_OP_ACCT_SRCH_DLG	Exists
OPAcctView	Account View	IDD_OP_ACCT_VIEW	Exists
OPAddrDlg	Address Dialog	IDD_OP_ADDR_DLG	Exists
OPAddrSrchDlg	Address Search Dialog	IDD_OP_ADDR_SRCH_DLG	Exists
OPAddrUpdtDlg	Address Update Dialog	IDD_OP_ADDR_UPDT_DLG	Exists
OPAtrbGridDlg	Attribute Grid Dialog	IDD_OP_ATRB_GRID_DLG	Exists
OPAvailSrvcDlg	Available Services Dialog	IDD_OP_AVAIL_SRVC_DLG	Exists
OPChrgGridDlg	Charge Grid Dialog	IDD_OP_CHRG_GRID_DLG	Exists
OPCmmntDlg	Comment Dialog	IDD_OP_CMMNT_DLG	New
OPCmmntUpdtDlg	Comment Update Dialog	IDD_OP_CMMNT_UPDT_DLG	Exists
OPCompHierView	Component Hierarchy Dialog	IDD_OP_COMP_HIER_VIEW	Exists
OPDscnctDlg	Disconnect Dialog	IDD_OP_DSCNCT_DLG	Exists
OPOrdHoldDlg	Order Hold Dialog	IDD_OP_ORDER_HOLD_DLG	New
OPRemitAddrDlg	Remittance Address Dialog	IDD_OP_REMIT_ADDR_DLG	Exists
OPRemitInfoDlg	Remittance Information Dialog	IDD_OP_REMIT_INFO_DLG	Exists
OPRemitMainDlg	Remittance Main Dialog	IDD_OP_REMIT_MAIN_DLG	Exists
OPSrvcDlg	Service Dialog	IDD_OP_SRVC_DLG	Exists
System Maintenance			
SMLogonDlg	Logon Dialog	IDD_SM_LOGON_DLG	Exists

Chapter

8

Tuxedo Data Services

SERVICE NAME	PARAMETER	ACTIVE SQL	ORDER SQL
All Accounts by Account Name opacctsbynm	ACCT_NAME	FROM ACCT_AC WHERE ACCT_NAME LIKE :ACCT_NAME_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ACCT_ID;	FROM ACCT_OP A, ORDER_OP B WHERE ACCT_NAME LIKE :ACCT_NAME_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY ACCT_ID;
All Accounts by Acct ID opacctsbyid	ACCT_ID	FROM ACCT_AC WHERE ACCT_ID >= :ACCT_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ACCT_ID;	FROM ACCT_OP A, ORDER_OP B WHERE A.ACCT_ID >= :ACCT_ID_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY A.ACCT_ID;
Accounts for a Specific Headquarter Account opacctsforhq	HQ_ACCT_ID	FROM ACCT_AC WHERE HQ_ACCT_ID = :ACCT_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ACCT_ID;	FROM ACCT_OP A, ORDER_OP B WHERE HQ_ACCT_ID = :ACCT_ID_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY A.ACCT_ID;
Account by Telephone Number opacctsbyphone	PHONE_NUM	FROM ATTR_AC A, COMP_AC B, ACCT_AC C WHERE A.CHAR_VAL LIKE :PHONE_NUM_FILTER AND A.ATTR_FMT = 'P' AND A.RCRD_EFF_END_DT = '12/31/9999 00:00:00'	FROM ATTR_OP A, COMP_OP B, ACCT_OP C, ORDER_OP D WHERE A.CHAR_VAL LIKE :PHONE_NUM_FILTER AND A.ATTR_FMT = 'P' AND A.ASN_COMP_ID = B.ASN_COMP_ID

SERVICE NAME	PARAMETER	ACTIVE SQL	ORDER SQL
		AND A.ASN_COMP_ID = B.ASN_COMP_ID AND B.RCRD_EFF_END_DT = A.RCRD_EFF_END_DT AND B.ACCT_ID = C.ACCT_ID AND C.RCRD_EFF_END_DT = B.RCRD_EFF_END_DT;	AND B.ACCT_ID = C.ACCT_ID AND C.ORDER_ID = C.MOST_RCNT_ORDER_ID AND C.ORDER_ID = D.ORDER_ID AND D.ORDER_STAT_CD = 'I';
Retrieve Account Information for a Specific opacctforid	ACCT_ID	FROM ACCT_AC WHERE ACCT_ID = :ACCT_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ACCT_ID;	FROM ACCT_OP A, ORDER_OP B WHERE A.ACCT_ID = :ACCT_ID_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY A.ACCT_ID;
HQ Accounts by Account Name *ophqacctsbynm*	ACCT_NAME	FROM ACCT_AC WHERE ACCT_NAME LIKE :ACCT_NAME_FILTER AND ACCT_ID = HQ_ACCT_ID AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ACCT_ID;	FROM ACCT_OP A, ORDER_OP B WHERE ACCT_NAME LIKE :ACCT_NAME_FILTER AND A.ACCT_ID = HQ_ACCT_ID AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY ACCT_ID;
HQ Accounts by Account ID *ophqacctsbyid*	ACCT_ID	FROM ACCT_AC WHERE ACCT_ID >= :ACCT_ID_FILTER AND ACCT_ID = HQ_ACCT_ID AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ACCT_ID;	FROM ACCT_OP A, ORDER_OP B WHERE A.ACCT_ID >= :ACCT_ID_FILTER AND A.ACCT_ID = HQ_ACCT_ID AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY A.ACCT_ID;
Account by Telephone Number	PHONE_NUM	FROM ATTR_AC A, COMP_AC B, ACCT_AC C WHERE A.CHAR_VAL LIKE :PHONE_NUM_FILTER	FROM ATTR_OP A, COMP_OP B, ACCT_OP C, ORDER_OP D WHERE A.CHAR_VAL LIKE :PHONE_NUM_FILTER

SERVICE NAME	PARAMETER	ACTIVE SQL	ORDER SQL
ophqacctbyphn		AND A.ATTR_FMT = 'P' AND A.RCRD_EFF_END_DT = '12/31/9999 00:00:00' AND A.ASN_COMP_ID = B.ASN_COMP_ID AND B.RCRD_EFF_END_DT = A.RCRD_EFF_END_DT AND B.ACCT_ID = C.ACCT_ID AND C.ACCT_ID = C.HQ_ACCT_ID AND C.RCRD_EFF_END_DT = B.RCRD_EFF_END_DT	AND A.ATTR_FMT = 'P' AND A.ASN_COMP_ID = B.ASN_COMP_ID AND B.ACCT_ID = C.ACCT_ID AND C.ACCT_ID = C.HQ_ACCT_ID AND C.ORDER_ID = C.MOST_RCNT_ORDER_ID AND C.ORDER_ID = D.ORDER_ID AND D.ORDER_STAT_CD = 'I';
All Addresses for a Headquarter Account *opaddrsforhq*	HQ_ACCT_ID	FROM ADDR_AC WHERE HQ_ACCT_ID = :HQ_ACCT_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ADDR_ID;	FROM ADDR_OP A, ORDER_OP B WHERE A.HQ_ACCT_ID = :HQ_ACCT_ID_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY ADDR_ID;
Address for an Account *opaddrforacct*	ADDR_ID	First retrieve ACCT_ADDR_ID from ACCT_OP or ACCT_AC FROM ADDR_AC WHERE ADDR_ID = :ADDR_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ADDR_ID;	First retrieve ACCT_ADDR_ID from ACCT_OP or ACCT_AC FROM ADDR_OP A, ORDER_OP B WHERE A.ADDR_ID = :ADDR_ID_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY ADDR_ID;
Comments for an Account *opacctcmmnt*	ACCT_ID	Select * From ACCT_CMMNT_AC Where ACCT_ID = ACCT_ID_FILTER And RCRD_EFF_END_DT = '12/31/9999'	

SERVICE NAME	PARAMETER	ACTIVE SQL	ORDER SQL
Assigned Services *opasnsrvcs*	ACCT_ID	FROM SRVC_AC WHERE ACCT_ID = :ACCT_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ASN_SRVC_ID;	FROM SRVC_OP A, ORDER_OP B WHERE A.ACCT_ID =:ACCT_ID_FILTER AND A.ORDER_ID = MOST_RCNT_ORDER_ID AND A.ORDER_ID = B.ORDER_ID AND ORDER_STAT_CD = 'I' ORDER BY ASN_SRVC_ID;
Assigned Components *opasncomps*	ASN_SRVC_ID ORDER_ID	FROM COMP_AC WHERE ASN_SRVC_ID = :ASN_SRVC_ID_FILTER AND RCRD_EFF_END_DT='12/31/9999 00:00:00' ORDER BY ASN_COMP_ID;	FROM COMP_OP WHERE ASN_SRVC_ID =:ASN_SRVC_ID_FILTER AND ORDER_ID = :ORDER_ID_FILTER ORDER BY ASN_COMP_ID;
Assigned Attribute *opasnattrs*	ORDER_ID	FROM ATTR_AC A, COMP_AC B WHERE B.ASN_SRVC_ID = :ASN_SRVC_ID_FILTER AND B.RCRD_EFF_END_DT='12/31/9999 00:00:00' AND B.ASN_COMP_ID = A.ASN_COMP_ID AND B.RCRD_EFF_END_DT = A.RCRD_EFF_END_DT ORDER BY A.ASN_COMP_ID, SEQ_NUM;	FROM ATTR_OP A, COMP_OP B WHERE B.ASN_SRVC_ID = :ASN_SRVC_ID_FILTER AND B.ORDER_ID = :ORDER_ID_FILTER AND B.ORDER_ID = A.ORDER_ID AND B.ASN_COMP_ID = A.ASN_COMP_ID ORDER BY A.ASN_COMP_ID, SEQ_NUM;
Assigned Charges *opasnchrgs*	ORDER_ID	FROM COMP_AC A, CHRG_AC B WHERE A.ASN_SRVC_ID = :ASN_SRVC_ID_FILTER AND A.RCRD_EFF_END_DT='12/31/9999 00:00:00' AND A.ASN_COMP_ID = B.ASN_COMP_ID AND A.RCRD_EFF_END_DT = B.RCRD_EFF_END_DT ORDER BY CHRG_ID;	FROM COMP_OP A, CHRG_OP B WHERE A.ASN_SRVC_ID = :ASN_SRVC_ID_FILTER AND A.ORDER_ID = :ORDER_ID_FILTER AND A.ORDER_ID = B.ORDER_ID AND A.ASN_COMP_ID = B.ASN_COMP_ID ORDER BY CHRG_ID;
Insert Order opinsrtorder			

SERVICE NAME	PARAMETER	ACTIVE SQL	ORDER SQL
Insert Account opinsrtacct			
Insert Address opinsrtaddr			
Insert Assigned Services opinsrtsrvcs			
Insert Assigned Components opinsrtcomps			
Insert Assigned Attributes opinsrtattrs			
Insert Assigned Charges opinsrtchrgs			
Insert Comment opinsrtcmmnts			

Chapter
9

Vi Reference

Overview

The UNIX-based *vi* editor is used in some of the exercises in the Online Development course and by most programmers at CBIS for editing code on the UNIX machine. This reference provides an overview of the some of the most common features of this editor.

Basic commands

To open a file in the *vi* editor, type **vi filename** at the command prompt.

To understand how *vi* works, you must know that there are two “modes” that you can be in: command mode and insert mode. When a file is first opened, you are in *command mode*. To enter command mode at any time, press the <Esc> key.

a	Append after the cursor
A	Append at the end of the line
c	Begin change operation
C	Change to end of line
i	Insert before cursor
I	Insert at beginning of line
o	Open a line below current line
O	Open a line above current line
R	Begin overwriting text
s	Substitute a character
S	Substitute an entire line

Movement commands

Character

h, j, k, l	Left, down, up, right
<Spacebar>	Right

Text

w, W, b, B	Forward, backward by word
e, E	End of word
), (	Beginning of next current sentence
}, {	Beginning of next, current paragraph
]], [[	Beginning of next, current section

Lines

0,$	First, last position of current line
^	First nonblank character of current line
+,-	First character of next, previous line

<Enter>	First character of next line
n|	Column *n* of current line
H	Top line of screen
M	Middle line of screen
L	Last line of screen
*n*H	*n* lines after top line
*n*L	*n* lines from last line

Screens

Ctrl-F	Scroll forward one screen
Ctrl-B	Scroll backward one screen
Ctrl-D	Scroll down one-half screen
Ctrl-U	Scroll up one-half screen
Ctrl-E	Show one more line at bottom of window
Ctrl-Y	Show one more line at top of window
z RETURN	Reposition line with cursor: to top of screen
Ctrl-L	Redraw screen
Ctrl-N	One line down

Searches

/*test*	Search forward for text
n	Repeat previous search
N	Repeat search in opposite direction
/	Repeat forward search
?	Repeat previous search backward
? *text*	Search backward for *text*
/*text*/+*n*	Goto line *n* after *text*
/*text*?-*n*	Goto line *n* before *text*
%	Find match of current ( ), { }, or [ ].
f*x*	Move search forward to *x* on current line
F*x*	Move search backward to *x* on current line
t*x*	Search forward before *x* on current line
T*x*	Search back after *x* on current line
,	Reverse search direction of last f, F, t, T
;	Repeat last character search (f, F, t, T)

Line numbering

Ctrl-G	Display current line number
*n*G	Move to line number *n*
G	Move to last line in file
:*n*	Move to line number *n*

Marking position

m*x*	Mark current position with character *x*
‘*x*	Move cursor to mark *x*
‘*x*	Move to start of line containing *x*
“	Return to previous mark (or to location prior to search)
`’	Like above, but return to start of line

Edit commands

Remember that *c*, *d*, and *y* are basic editing operators.

Inserting new text

a	Append after the cursor
A	Append to the end of the line
i	Insert before cursor
I	Insert at the beginning of the line
o	Open a line below the cursor
O	Open a line above the cursor
Esc	Terminate insert mode
Ctrl-J	Move down one line
Enter	Move down one line
Ctrl-I	Insert a tab
Ctrl-T	Move to next tab setting
Backspace	Move back one character
Ctrl-H	Move back one character
Ctrl-U	Delete current line
Ctrl-V	Quote next character
Ctrl-W	Move back one word

Changing and deleting text

cw	Change word
cc	Change line
C	Change text from current position to end of line
dd	Delete current line
*n*dd	Delete *n* lines
D	Delete remainder of line
dw	Delete word
d}	Delete up to next paragraph
d^	Delete back to beginning of line
d/*pat*	Delete up to first occurrence of pattern
dn	Delete up to next occurrence of pattern

Saving and Exiting

ZZ	Quitting vi, writing the file only if changes were made
:x	Same as ZZ
:wq	Write and quit file
:w	Write file
:w *file*	Save copy to *file*
:*n1*, *n2*w *file*	Write line *n1* to *n2* to new file
:*n1, n2w>>file*	Append lines *n1* to *n2* to existing *file*
:w!	Write file (overriding protection)
:w! *file*	Overwrite *file* with current buffer
:w %.*new*	Write current buffer named *file* as *file.new*
:q	Quit file
:q!	Quit file (discarding edits)
Q	Quit vi and invoke ex
:vi	Return to vi after Q command
:e *file2*	Edit *file2* without leaving vi
:n	Edit next file
:e!	Return to version of file at time of last write
:e#	Edit alternate file
%	Current filename
#	Alternate filename

df*a*	Delete up to and including *a* on current line
dt*a*	Delete up to (not including) *a* on current line
dL	Delete up to last line on screen
dG	Delete to end of line
p	Insert last deleted text after cursor
P	Insert last deleted text before cursor
r*x*	Replace character with *x*
R*text*	Replace *text* beginning at cursor
s	Substitute character
4s	Substitute four characters
S	Substitute entire line
u	Undo last change
U	Restore current line
x	Delete current cursor position
X	Delete back one character
5X	Delete previous five characters
.	Repeat last change
~	Reverse case

Copying and Moving

Y	Copy current line
yy	Copy current line
"*x*yy	Yank current line to buffer *x*
"*x*dd	Delete into buffer *x*
"*X*d	Delete and append into buffer *x*
"*x*p	Put contents of buffer *x*
y]]	Copy to next section heading
ye	Copy to end of word

Accessing Multiple Files

:e *file*	Edit another *file*, current file becomes alternate
:e!	Restore last saved version of current file
:e + *file*	Begin editing at end of file
:e + *n file*	Open *file* at line *n*

:e#	Open to previous position in alternate file
:ta *tag*	Edit file at location *tag*
:n	Edit next file
:n!	Forces next file
:n *files*	Specify new list of *files*
Ctrl-G	Show current file and line number
:args	Display multiple files to be edited
:rew	Rewind list of multiple files to top

Interacting with UNIX

:r *file*	Read in contents of *file* after cursor
:r !*command*	Read in output from *command* after current line
:*n*r !*command*	Like above, but place after line *n* (0 for top of file)
:!*command*	Run command, then return
!*object command*	Send buffer *object* to UNIX *command*; replace with output
:*n1,n2*!*command*	Send lines *n1 –n2* to *command*; replace with output
n!!*command*	Send *n* lines to UNIX *command*; replace with output.
!!	Repeat last system command.
:sh	Create subshell; return to file with *EOF*
Ctrl-Z	Suspend editor; resume with fg
:so *file*	Read and execute commands from *file*

Macros

:ab *in out*	Use *in* as abbreviation for *out*
:unab *in*	Remove abbreviation for *in*
:ab	List abbreviations
:map *c sequence*	Map character *c* as *sequence* of commands
:unmap *c*	Disable map for *c*
:map	List characters that are mapped
:map! *c sequence*	Map character *c* to input mode *sequence*
:unmap! *c*	Disable input mode map
:map!	List characters that are mapped to input mode

The following keys can be mapped

Letters:	g K q V v
Control Keys:	^A ^K ^O ^T ^W ^X
Symbols	- * \

Miscellaneous Commands

J	Join two lines
:j!	Join two lines, preserving blank spaces
<<	Shift this line left one shift width (default is 8 spaces)
>>	Shift this line right one shift width (default is 8 space)
>}	Shift right to end of paragraph
<%	Shift left until matching ( ), { }, [ ], etc. (Cursor must be on the matching symbol